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

June 29, 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 dato che per le stringhe il tutto diventerebbe solo una inutile ripetizione dello stesso principio.

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.
Per l'insieme vuoto do la possibilita' all'utente sia di scegliere a priori che l'insieme deve essere vuoto sia di poterlo immettere durante la creazione dell'insieme.

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

4.1 Programma

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
17 typedef struct Operazione
18 {
19   double   *operando_a;
20   double   *operando_b;
21   double   *risultati;
22
23 } operazione_t;
24
25 typedef struct RelBin
26 {
27   /* coppia numerica */
28
29   double  *primo_termine;
30   double *secondo_termine;
31
32   /* variabile per sapere il numero delle coppie */
33
34   int dimensione;
35 } rel_bin;
36
37 typedef struct Insieme
38 {
39   double* elementi_insieme;
40   int numero_elementi;
41 } insieme_t;
42
43 /*********************************/
44 /* dichiarazione delle funzioni */
```

```
45 /*******************************/
46
47 void errore(void);
48 void svuota_buffer(void);
49 insieme_t acquisisci_insieme(void);
50 void stampa_insieme(insieme_t);
51 insieme_t crea_insieme_vuoto(void);
52 rel_bin acquisisci_rel_bin(insieme_t);
53 int acquisisci_elemento(insieme_t);
54 void stampa_rel_bin(rel_bin);
55 operazione_t acquisisci_operazione(insieme_t);
56 int controllo_simmetria(rel_bin);
57 int controllo_riflessivita(rel_bin);
58 int controllo_transitivita(rel_bin);
59 int relazione_equivalenza(rel_bin);
60 int controllo_chiusura(insieme_t,
61                        operazione_t);
62 void controllo_congruenza(rel_bin,
63                           insieme_t,
64                           operazione_t,
65                           int);
66 void informazioni_sul_programma(void);
67 void scelta_operazione(void);
68 int ripeti(void);
69
70 /*****************/
71 /* funzione main */
72 /*****************/
73
74 int main()
75 {
76   /*variabile per dare la possibilita'
77   all'utente di utilizzare il programma
78   ricorsivamente*/
79   int ripetere;
80   ripetere = 1;
81
82   informazioni_sul_programma();
83
84   while (ripetere == 1)
85   {
86
87     scelta_operazione();
88     ripetere = ripeti();
89
90   }
91
92   return 0;
93 }
```

```
94
95  /*funzione per poter riportare un segnale di errore dopo un
        acquisizione da tastiera errata*/
96  void errore()
97  {
98    printf("\n\n Hai inserito un valore errato.");
99    printf("\n Inserire un valore corretto: ");
100
101   return;
102 }
103
104 /*funzione per poter pulire il buffer*/
105 void svuota_buffer()
106 {
107   /*variabile per svuotare il buffer*/
108   char carattere_non_letto;
109   do
110   {
111     carattere_non_letto = getchar();
112   }
113   while (carattere_non_letto != '\n');
114
115   return;
116 }
117
118 /*informazioni sul programma*/
119 void informazioni_sul_programma()
120 {
121   printf("\n ************************************");
122   printf("************************\n");
123   printf("\n Questo programma acquisisce nel seguente");
124   printf(" ordine:\n");
125   printf("\n 1) Un insieme;\n 2) Una relazione binaria su ");
126   printf("quell'insieme;\n 3) Un'operazione binaria su quell");
127   printf("'insieme.\n\n Poi verifica se l'insieme e' chiuso ");
128   printf("rispetto all'operazione \n ");
129   printf(" e se la relazione e' una");
130   printf(" congruenza rispetto all'operazione.\n");
131
132   return;
133 }
134
135 /*Funzione per chiedere all'utente cosa vuole fare*/
136 void scelta_operazione()
137 {
138   /*variabile per il controllo della scelta*/
139   int scelta;
140   /*variabile per controllare che la
141   lettura sia avvenuta correttamente*/
```

```
142   int lettura_effettuata;
143   /*variabile per il salvataggio del
144   risultato della verifica della chiusura*/
145   int chiusura;
146
147   /* variabili per insieme, relazione
148   e operazione*/
149   operazione_t operazione;
150   insieme_t insieme;
151   rel_bin relazione;
152
153   /*inizializzo le variabili*/
154   scelta = 0;
155   lettura_effettuata = 0;
156   chiusura = 1;
157
158   printf("\n **********************************");
159   printf("*****************************\n");
160   printf("\n\n Digitare:\n 1 - se si vuole iniziare con");
161   printf(" l'acquisizione dell'insieme,\n 2 - se si vuole ");
162   printf("inserire l'insieme vuoto,");
163   printf("\n 3 - se si vogliono avere ");
164   printf("informazioni sul programma ");
165   printf("\n 4 - uscire da questo menu': ");
166
167   do
168   {
169     lettura_effettuata = scanf("%d",&scelta);
170     if ((lettura_effettuata != 1)
171       || (scelta != 1 && scelta != 2
172       && scelta != 3 && scelta != 4)
173       || (lettura_effettuata != 1))
174     {
175       errore();
176       svuota_buffer();
177       scelta=5;
178     }
179
180   }while ((scelta != 1 && scelta != 2
181          && scelta != 3 && scelta != 4) || lettura_effettuata != 1);
182
183   if (scelta == 1)
184   {
185
186     insieme = acquisisci_insieme();
187     stampa_insieme(insieme);
188     if (insieme.numero_elementi != 0)
189     {
190       relazione = acquisisci_rel_bin(insieme);
```

```
191       stampa_rel_bin(relazione);
192       operazione = acquisisci_operazione(insieme);
193       chiusura = controllo_chiusura(insieme,
194                                     operazione);
195       controllo_congruenza(relazione,
196                            insieme,
197                            operazione,
198                            chiusura);
199     }
200   }
201   if (scelta == 2 || insieme.numero_elementi == 0)
202   {
203
204     printf("\n\n ************* INSIEME");
205     printf(" VUOTO **********************\n");
206     insieme = crea_insieme_vuoto();
207     printf("\n L'insieme che si e' scelto e' vuoto,");
208     printf(" quindi qualsiasi \n sia la relazione");
209     printf(", simmetria, riflessivita' e transitivita'\n");
210     printf(" sono sempre verificate.\n Per convenzione ");
211     printf("diciamo anche che qualsiasi sia\n l'operazione");
212     printf(" e' chiusa rispetto all'insieme");
213   }
214
215   if (scelta == 3)
216     informazioni_sul_programma();
217
218   return;
219 }
220
221
222 /*Funzione per ripetere il procedimento*/
223 int ripeti()
224 {
225   int ripetere;
226   int lettura_effettuata;
227
228   ripetere = 0;
229
230   printf("\n\n Digitare:\n 1 - se si vuole eseguire");
231   printf(" un altra operazione,");
232   printf("\n 2 - se si vuole terminare il programma: ");
233   do
234   {
235     lettura_effettuata = scanf("%d",&ripetere);
236     if (lettura_effettuata != 1 || (ripetere != 1 && ripetere != 2))
237     {
238       errore();
239       svuota_buffer();
```

```
240     }
241   }
242   while (lettura_effettuata != 1 || (ripetere != 1 && ripetere != 2)
          );
243
244
245   return (ripetere);
246 }
247
248
249 /*****************************/
250 /* acquisizione dell'insieme */
251 /*****************************/
252
253 insieme_t acquisisci_insieme()
254 {
255   /*dichiaro la struttura insieme*/
256
257   insieme_t insieme;
258
259   /*variabile contatore */
260   int  i;
261   /*variabile per il controllo di doppioni*/
262   int buffer_vuoto;
263   /*variabile contatore*/
264   int  j;
265   /*variabile per controllare la fine dell acquisizione*/
266   int controllo;
267   /*variabile per terminare l'acquisizione*/
268   int  finisci_di_acquisire;
269   /*variabile per verificare che la
270   acquisizione vada a buon fine*/
271   int  elemento_acquisito;
272   /*variabile necessaria allo
273   svuotamento del buffer*/
274   char carattere_non_letto;
275   /*variabile per acquisire ogni
276   elemento temporaneamente*/
277   double temporaneo;
278
279   /*inizializzo le variabili*/
280   elemento_acquisito = 0;
281   j = 0;
282   i = 0;
283   temporaneo = 0;
284   insieme.numero_elementi = 50;
285   finisci_di_acquisire = 0;
286   controllo = 0;
287   buffer_vuoto = 1;
```

```
288  /*alloco memoria*/
289  insieme.elementi_insieme = (double *)
290                          malloc (insieme.numero_elementi);
291
292  /*inizio la vera e propria acquisizione*/
293
294  printf("\n\n Si e' scelto di acquisire un'insieme\n");
295
296  /*chiedo se l'utente vuole inserire lo 0*/
297
298  printf("\n\n ************** ACQUISIZIONE DELL'");
299  printf("INSIEME **********************\n");
300
301  printf("\n\n Per terminare l'acquisizione digitare a\n\n");
302
303  while (finisci_di_acquisire != 1)
304  {
305    controllo = 0;
306    insieme.elementi_insieme = (double *)
307                            realloc (insieme.elementi_insieme,
308                                    (i+1) * sizeof (double));
309    printf("\n Digitare ora il %d elemento: ",i+1);
310    /*svuoto il buffer prima di acquisire*/
311    if (buffer_vuoto != 1)
312      svuota_buffer();
313
314    buffer_vuoto = 0;
315    elemento_acquisito = scanf("%lf",&temporaneo);
316    if (i >= 0 && finisci_di_acquisire != 1 )
317      insieme.elementi_insieme[i] = temporaneo;
318
319    /*controllo se c'e' stato un errore nell'acquisizione*/
320    if (elemento_acquisito != 1)
321    {
322      do
323      {
324        carattere_non_letto = getchar();
325        /*controllo se l utente ha digitato il carattere di
             terminazione*/
326        if ((carattere_non_letto == 'a') && (controllo == 0))
327        {
328          finisci_di_acquisire = 1;
329          insieme.numero_elementi = i;
330        }
331        /*controllo che mi permette di verificare se e' stato
             digitato solo un carattere */
332        if (controllo > 1)
333          finisci_di_acquisire = 0;
334        controllo++;
```

```
335
336       }
337       while (carattere_non_letto != '\n');
338       /*mi segno che il buffer e' gia' vuoto per non andarlo a
              svuotare una seconda volta*/
339       buffer_vuoto = 1;
340       i--;
341     }
342
343     /*controllo che l'elemento non sia gia presente nell insieme*/
344
345     for (j = i-1; j >= 0; j--)
346     {
347       if (temporaneo == insieme.elementi_insieme[j])
348       {
349         i--;
350         j = 0;
351       }
352     }
353     i++;
354   }
355
356   return (insieme);
357 }
358
359 /***********************/
360 /* stampa dell'insieme */
361 /***********************/
362
363 void stampa_insieme(insieme_t insieme)
364 {
365   /*variabile contatore*/
366   int i;
367
368   printf("\n\n ************** STAMPA DELL'");
369   printf("INSIEME **************************\n");
370   printf("\n\n L'insieme acquisito e':");
371   printf("\n\n { ");
372   i=0;
373
374   while (i < insieme.numero_elementi)
375   {
376     printf("%.2lf",insieme.elementi_insieme[i]);
377     if (i+1 < insieme.numero_elementi)
378       printf(" ; ");
379     i++;
380   }
381   printf(" }\n\n");
382
```

```
383   return;
384 }
385
386 insieme_t crea_insieme_vuoto()
387 {
388   /*variabile per la struttura insieme*/
389   insieme_t insieme;
390
391   insieme.elementi_insieme = (double *) malloc (1);
392   insieme.numero_elementi = 0;
393   return (insieme);
394 }
395
396 /*Funzione che acquisisce la relazione binaria*/
397
398 rel_bin acquisisci_rel_bin(insieme_t insieme)
399 {
400
401   rel_bin relazione;
402
403   /*variabile utile ad uscire dal ciclo
404   di acquisizione*/
405   int acquisizione_finita,
406       /*variabile per il controllo
407       dell'acquisizione*/
408       risultato_lettura,
409       /*variabile contatore*/
410       i,
411       /*relazione vuota*/
412       relazione_vuota,
413       /*variabile per il primo termine*/
414       primo_termine_acquisito;
415
416   printf("\n\n ********* ACQUISIZIONE DELLA");
417   printf("RELAZIONE BINARIA **********\n");
418   printf("\n\n Si vuole acquisire una relazione vuota?");
419   printf("\n\n Digitare:\n 0 - si\n");
420   printf(" 1 - no: ");
421   do
422   {
423     risultato_lettura = scanf("%d",&relazione_vuota);
424     if ((risultato_lettura != 1 || relazione_vuota != 0) &&
          relazione_vuota != 1)
425     {
426       errore();
427       svuota_buffer();
428     }
429   }
```

```
430   while ((risultato_lettura != 1 || relazione_vuota != 0) &&
          relazione_vuota != 1);
431   if (relazione_vuota == 0)
432     printf(" si e' scelto di inserire una relazione vuota");
433
434   /*inizializzo le variabili*/
435   acquisizione_finita = 1;
436   primo_termine_acquisito = 0;
437   relazione.dimensione = 0;
438   /*alloco memoria*/
439   relazione.primo_termine = (double *) malloc (2);
440   relazione.secondo_termine = (double *) malloc (2);
441   if (relazione_vuota == 1)
442   {
443     while (acquisizione_finita == 1)
444     {
445       primo_termine_acquisito = 0;
446       relazione.dimensione++;
447       acquisizione_finita = 2;
448
449       /*Acquisisco i termini della coppia*/
450
451       printf ("\n\n Inserisci i termini della coppia \n ");
452
453       relazione.primo_termine = (double *)
454                               realloc (relazione.primo_termine,
455                                       (relazione.dimensione+1)
456                                       * sizeof (double));
457
458       relazione.secondo_termine = (double *)
459                                 realloc (relazione.secondo_termine,
460                                         (relazione.dimensione+1)
461                                         * sizeof (double));
462       risultato_lettura = 0;
463
464
465       /*Acquisisco il primo termine*/
466       if (primo_termine_acquisito == 0)
467       {
468         printf (" Primo Termine: ");
469         relazione.primo_termine[relazione.dimensione - 1] =
470           acquisisci_elemento(insieme);
471       }
472       primo_termine_acquisito = 1;
473
474       /*Acquisisco il secondo termine*/
475       if (primo_termine_acquisito == 1)
476       {
477         printf (" Secondo Termine: ");
```

```
478         relazione.secondo_termine[relazione.dimensione - 1]
479           = acquisisci_elemento(insieme);
480
481         for (i=relazione.dimensione-2; i>=0; i--)
482           if (relazione.primo_termine[relazione.dimensione - 1]
483               == relazione.primo_termine[i])
484             if (relazione.secondo_termine[relazione.dimensione -1]
485                 == relazione.secondo_termine[i])
486             {
487               relazione.dimensione--;
488               i = 0;
489             }
490       }
491
492       /*Chiedo all'utente se ci sono altre coppie*/
493
494       do
495       {
496         printf("\n\n Digitare:\n 0 - per");
497         printf(" terminare l'acquisizione,");
498         printf("\n 1 - se si vuole acquisire un altra coppia: ");
499         risultato_lettura = scanf ("%d",
500                                    &acquisizione_finita);
501         if (acquisizione_finita < 0 ||
502             acquisizione_finita > 1 || risultato_lettura != 1)
503         {
504           errore();
505           svuota_buffer();
506         }
507       }
508       while (acquisizione_finita < 0 || acquisizione_finita > 1 );
509     }
510   }
511   svuota_buffer();
512   return (relazione);
513 }
514
515 /************FUNZIONE DI STAMPA****************/
516
517 void stampa_rel_bin(rel_bin stampa)
518 {
519   /*variabile contatore*/
520   int i = 0;
521
522   printf("\n\n ********* STAMPA DELLA RELAZIONE BINARIA *****");
523   printf ("****************\n\n La relazione binaria e':");
524   printf ("\n\n {");
525
526   /******Stampa per coppie numeriche *****/
```

```
527
528   while (i < stampa.dimensione)
529   {
530     printf ("(%.2lf,%.2lf)",
531             stampa.primo_termine[i],
532             stampa.secondo_termine[i]);
533     if (i+1 != stampa.dimensione)
534       printf (" ; ");
535     i++;
536   }
537   printf("}\n");
538   return;
539 }
540
541 int acquisisci_elemento(insieme_t insieme)
542 {
543   /*variabile per il controllare che
544   gli elementi acquisiti siano stati
545   letti correttamente*/
546   int lettura_corretta,
547       /*variabile contatore*/
548       i,
549       /*variabile di controllo per verificare
550       la non ripetizione di elementi*/
551       elemento_trovato;
552
553   double elemento;
554   /* inizializzo le variabili */
555   elemento = 0;
556   lettura_corretta = 1;
557
558   do
559   {
560     /* controllo che i valori siano
561         stati letti correttamente
562      e nel caso svuoto il buffer */
563
564     if (lettura_corretta != 1)
565     {
566       svuota_buffer();
567       printf ("\n Verificare che l'elemento");
568       printf (" appartenga \n all'insieme");
569       printf (" precedentemente inserito. \n ");
570       errore();
571     }
572     lettura_corretta = scanf("%lf",&elemento);
573
574     /* verifico se l'elemento che si
575        vuole utilizzare nella relazione
```

```
576       e' presente nell'insieme inserito */
577
578     elemento_trovato = 0;
579
580     for (i=0; i < insieme.numero_elementi; i++)
581       if (elemento == insieme.elementi_insieme[i])
582         elemento_trovato = 1;
583
584     if (elemento_trovato == 0)
585       lettura_corretta = 0;
586   }
587   while (lettura_corretta == 0);
588
589   svuota_buffer();
590   return (elemento);
591 }
592
593
594 /* Acquisisco l'operazione*/
595
596 operazione_t acquisisci_operazione(insieme_t insieme)
597 {
598   /*variabile per acquisire l operazione*/
599   operazione_t operazione;
600   /*variabile contatore*/
601   int i,
602       j,
603       /*variabile che stabilisce la dimensione dell'array*/
604       dimensione,
605       /*variabile per il controllo*/
606       controllo;
607
608   i = 0;
609   j = 0;
610   dimensione = 0;
611
612   operazione.risultati = (double *) malloc (2);
613   operazione.operando_a = (double *) malloc (2);
614   operazione.operando_b = (double *) malloc (2);
615   printf("\n\n ********* ACQUISIZIONE ");
616   printf("DELL'OPERAZIONE **********************\n");
617   printf(" \n\n Inserire ora i risultati dell'operazioni: \n");
618   printf(" \n Digitare 999 per risultati ");
619   printf("impossibili o indeterminati. \n");
620
621   for (i = 0; i < insieme.numero_elementi; i++)
622   {
623     for (j = 0; j < insieme.numero_elementi; j++)
624     {
```

```
625       operazione.risultati = (double *)
626                            realloc (operazione.risultati,
627                                     (dimensione+1)
628                                     * sizeof (double));
629       operazione.operando_a = (double *)
630                             realloc (operazione.operando_a,
631                                      (dimensione+1)
632                                      * sizeof (double));
633       operazione.operando_b = (double *)
634                             realloc (operazione.operando_b,
635                                      (dimensione+1)
636                                      * sizeof (double));
637       operazione.operando_a[dimensione] = insieme.elementi_insieme[i
              ];
638       operazione.operando_b[dimensione] = insieme.elementi_insieme[j
              ];
639       printf("\n %f * %f = ",insieme.elementi_insieme[i],
640              insieme.elementi_insieme[j]);
641       do
642       {
643
644         controllo = scanf("%lf",
645                           &operazione.risultati[dimensione]);
646         if (controllo != 1)
647         {
648           errore();
649           svuota_buffer();
650         }
651       }
652
653       while (controllo != 1);
654       dimensione++;
655     }
656   }
657   svuota_buffer();
658   return (operazione);
659 }
660
661 int controllo_chiusura(insieme_t insieme,
662                        operazione_t operazione)
663 {
664   int i,
665       j,
666       chiusura;
667
668   i = 0;
669   j = 0;
670   chiusura = 0;
671
```

```
672   for (i = 0;
673        i<(insieme.numero_elementi*insieme.numero_elementi);
674        i++)
675   {
676     chiusura = 0;
677     if (operazione.risultati[i] != 999)
678       for (j=0; j<insieme.numero_elementi; j++)
679         if (operazione.risultati[i] ==
680             insieme.elementi_insieme[j])
681         {
682           chiusura = 1;
683           j = insieme.numero_elementi+1;
684         }
685     if (chiusura == 0)
686       i = (insieme.numero_elementi*insieme.numero_elementi);
687   }
688   printf("\n\n ************ CHIUSURA **********************");
689   printf("******************\n");
690   if (chiusura == 0)
691     printf("\n\n La chiusura non e' verificata\n");
692   if (chiusura == 1)
693     printf("\n\n La chiusura e' verificata\n");
694
695   return (chiusura);
696 }
697
698 int controllo_riflessivita(rel_bin verifica)
699 {
700   /*variabile contatore*/
701   int i,
702       j,
703       k,
704       /*variabili per contare i riscontri*/
705       riscontro,
706       secondo_riscontro,
707       /*variabile per vedere se e' stata verificata la riflessivita
              '*/
708       riflessivita;
709
710   riflessivita = 1;
711   i = 0;
712   j = 0;
713   k = 0;
714   riscontro = 0;
715   secondo_riscontro = 0;
716
717   /*Verifica riflessivita'*/
718
719   /*Definizione: una relazione per la quale
```

```
 esiste almeno un elemento che non e'in relazione
con se' stesso non soddisfa la definizione di riflessivita'*/

while ( (i < verifica.dimensione) && (k < verifica.dimensione))
{

  /*Verifica riflessivita' per numeri*/

  riscontro = 0;
  secondo_riscontro = 0;
  if (verifica.primo_termine[i] == verifica.secondo_termine[i])
    riscontro++;/*Controllo se c'e' stato un riscontro a,a*/
  secondo_riscontro++;
  if (riscontro != 0)
  {
    i++;
    k++;
  }
  /**/
  else
  {
    j = 0;
    riscontro = 0;
    secondo_riscontro = 0;

    /* Controllo la riflessivita' per
    gli elementi del primo insieme */

    while (j < verifica.dimensione)
    {
      if (j == i)
        j++;
      else
      {
        if (verifica.primo_termine[i] ==
            verifica.primo_termine[j])
          if (verifica.primo_termine[j] ==
              verifica.secondo_termine[j])
            riscontro++;

        j++;
      }
    }

    j = 0;

    /*Controllo la riflessivita' per gli
    elementi del secondo insieme*/

```

```
769       while (j < verifica.dimensione)
770       {
771         if (j == k)
772           j++;
773         else
774         {
775           if (verifica.secondo_termine[k] ==
776               verifica.secondo_termine[j])
777             if (verifica.primo_termine[j] ==
778                 verifica.secondo_termine[j])
779               secondo_riscontro++;
780
781           j++;
782         }
783       }
784       if (riscontro != 0)
785         i++;
786
787       /**** Se non c'e' stato un riscontro di riflessivita'
788        esco e imposto la riflessivita' a 0 *****/
789
790       else
791       {
792         i = verifica.dimensione;
793         riflessivita = 0;
794       }
795
796       if (secondo_riscontro != 0)
797         k++;
798
799       else
800       {
801         k = verifica.dimensione;
802         riflessivita = 0;
803       }
804     }
805
806   }
807
808
809   /********** Fine riflessivita **********************/
810   return (riflessivita);
811 }
812
813 int controllo_transitivita(rel_bin verifica)
814 {
815   /*variabile contatore*/
816   int i,
817       j,
```

```
818     k,
819     /*variabile per controllare la transitivita'*/
820     transitivita;
821
822   /*IMPOSTO LA TRANSITIVITA' INIZIALMENTE COME VERA
823    E AZZERO I CONTATORI*/
824
825   transitivita = 1;
826   i = 0;
827   j = 0;
828   k = 0;
829
830   /*VERIFICA TRANSITIVITa' PER NUMERI*/
831
832
833   while (i < verifica.dimensione)
834   {
835     j = 0;
836
837     while (j < verifica.dimensione)
838     {
839       k = 0;
840
841       if (verifica.secondo_termine[i] ==
842           verifica.primo_termine[j])
843       {
844         transitivita = 0;
845
846         while (k < verifica.dimensione)
847         {
848           if (verifica.primo_termine[i] ==
849               verifica.primo_termine[k])
850           {
851             if (verifica.secondo_termine[k] ==
852                 verifica.secondo_termine[j])
853             {
854               transitivita = 1;
855               k = verifica.dimensione;
856             }
857           }
858
859           k++;
860         }
861
862         if (transitivita==0)
863         {
864           j = verifica.dimensione;
865           i = verifica.dimensione;
866         }
```

```
867       }
868
869       j++;
870     }
871
872     i++;
873   }
874
875   /************ Fine controllo Transitivita' ************/
876
877   return (transitivita);
878
879 }
880
881 int controllo_simmetria(rel_bin verifica)
882 {
883   /*variabili contatore*/
884   int i,
885       j,
886       /*variabile per controllare se c'e' stato un riscontro*/
887       riscontro,
888       /*variabile per controllare la simmetria*/
889       simmetria;
890
891   simmetria = 1;
892
893
894   i = 0;
895   j = 0;
896   riscontro = 0;
897
898   /*controllo della simmetria per numeri*/
899
900   while ( i < verifica.dimensione)
901   {
902
903     j = 0;
904     while ( j < verifica.dimensione)
905     {
906
907       if (verifica.primo_termine[i] ==
908           verifica.secondo_termine[j])
909         if (verifica.primo_termine[j] ==
910             verifica.secondo_termine[i])
911           riscontro++;
912       j++;
913     }
914
915     if (riscontro == 0)
```

```
916       {
917         j = verifica.dimensione;
918         i = verifica.dimensione;
919         simmetria = 0;
920       }
921       riscontro = 0;
922       i++;
923     }
924
925     return (simmetria);
926   }
927
928
929   int relazione_equivalenza(rel_bin verifica)
930   {
931
932     /*variabili per controllare le propieta' dell'equivalenza*/
933     int riflessivita,
934         simmetria,
935         transitivita,
936         equivalenza;
937
938     equivalenza = 0;
939     riflessivita = controllo_riflessivita(verifica);
940     simmetria = controllo_simmetria(verifica);
941     transitivita = controllo_transitivita(verifica);
942
943     if (riflessivita == 1 && simmetria == 1 && transitivita == 1)
944     {
945       printf ("\n E' una relazione di equivalenza\n");
946       equivalenza = 1;
947     }
948
949     if (riflessivita == 0)
950     {
951       printf ("\n Non e'una relazione di equivalenza ");
952       printf ("perche' non e' riflessiva\n");
953     }
954     if (simmetria == 0)
955     {
956       printf ("\n Non e'una relazione di equivalenza ");
957       printf ("perche' non e' simmetrica\n");
958     }
959     if (transitivita == 0)
960     {
961       printf ("\n Non e'una relazione di equivalenza ");
962       printf ("perche' non e' transitiva\n");
963     }
964     return (equivalenza);
```

```
965 }
966
967
968 void controllo_congruenza(rel_bin relazione,
969                           insieme_t insieme,
970                           operazione_t operazione,
971                           int chiusura)
972 {
973   printf("\n\n ********* CONTROLLO LA CONGRUENZA");
974   printf(" **************************\n");
975   /*variabile per il controllo dell'equivalenza*/
976   int equivalenza,
977       /*variabile di controllo*/
978       controllo,
979       /*variabili contatori*/
980       i,
981       j,
982       k;
983   if (relazione.dimensione != 0)
984     equivalenza = relazione_equivalenza(relazione);
985   else
986   {
987     printf("\n\n La relazione vuota ");
988     printf("non e' una relazione di equivalenza");
989     equivalenza = 0;
990   }
991   i = 0;
992   j = 0;
993   k = 0;
994   controllo = 1;
995
996   for (i = 0; i<relazione.dimensione; i++)
997   {
998     for (j = 0;
999          j<(insieme.numero_elementi*insieme.numero_elementi);
1000         j++)
1001    {
1002      if (relazione.primo_termine[i] ==
1003          operazione.operando_a[j])
1004        for (k = 0;
1005             k<(insieme.numero_elementi*insieme.numero_elementi);
1006             k++)
1007          if (relazione.secondo_termine[i] ==
1008              operazione.operando_a[k] &&
1009              operazione.operando_b[j] ==
1010              operazione.operando_b[k])
1011
1012            if (operazione.risultati[j]
1013                != operazione.risultati[k])
```

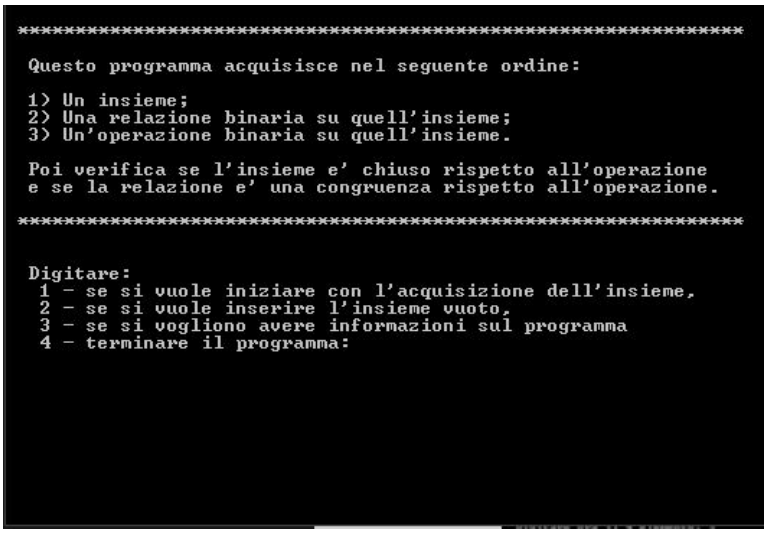
```
1014            {
1015              controllo = 0;
1016              k = (insieme.numero_elementi*
1017                   insieme.numero_elementi);
1018
1019              j = (insieme.numero_elementi*
1020                   insieme.numero_elementi);
1021
1022              i = relazione.dimensione;
1023            }
1024       if (relazione.primo_termine[i] ==
1025           operazione.operando_b[j])
1026         for (k = 0;
1027              k < insieme.numero_elementi*insieme.numero_elementi;
1028              k++)
1029           if (relazione.secondo_termine[i] ==
1030               operazione.operando_b[k] &&
1031               operazione.operando_a[j] ==
1032               operazione.operando_a[k])
1033
1034             if (operazione.risultati[j] !=
1035                 operazione.risultati[k])
1036             {
1037               controllo = 0;
1038               k = insieme.numero_elementi*
1039                    insieme.numero_elementi;
1040
1041               j = insieme.numero_elementi*
1042                    insieme.numero_elementi;
1043
1044               i = relazione.dimensione;
1045             }
1046     }
1047   }
1048
1049
1050   if (equivalenza == 0 || controllo == 0 || chiusura == 0)
1051     printf("\n\n La congruenza non e' verificata\n");
1052   else
1053     printf("\n\n La congruenza e' verificata\n");
1054
1055   return;
1056 }
```

4.2 Makefile

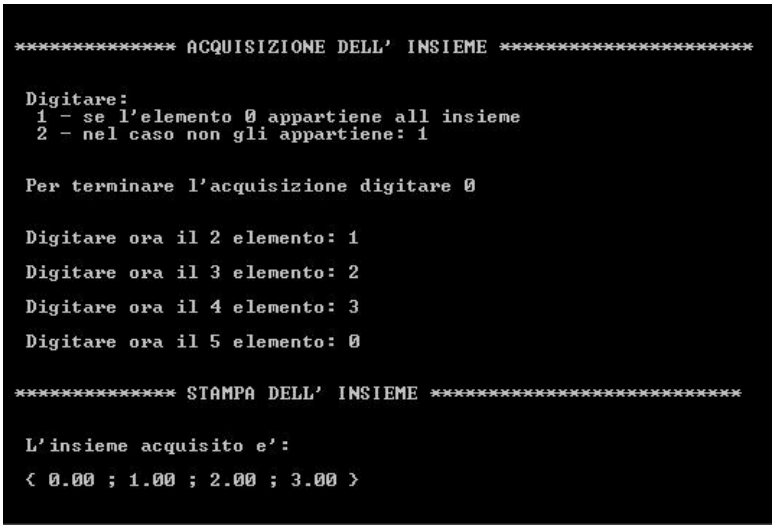
```
1 Progetto_sessione_estiva_PPL: Progetto_sessione_estiva_PPL.c
      Makefile
2   gcc -ansi -Wall -O Progetto_sessione_estiva_PPL.c -o
        Progetto_sessione_estiva_PPL
3 pulisci:
4   rm -f Progetto_sessione_estiva_PPL.o
5 pulisci_tutto:
6   rm -f Progetto_sessione_estiva_PPL Progetto_sessione_estiva_PPL.o
```

5 Testing del programma

Spiego all'utente cosa fa il programma.

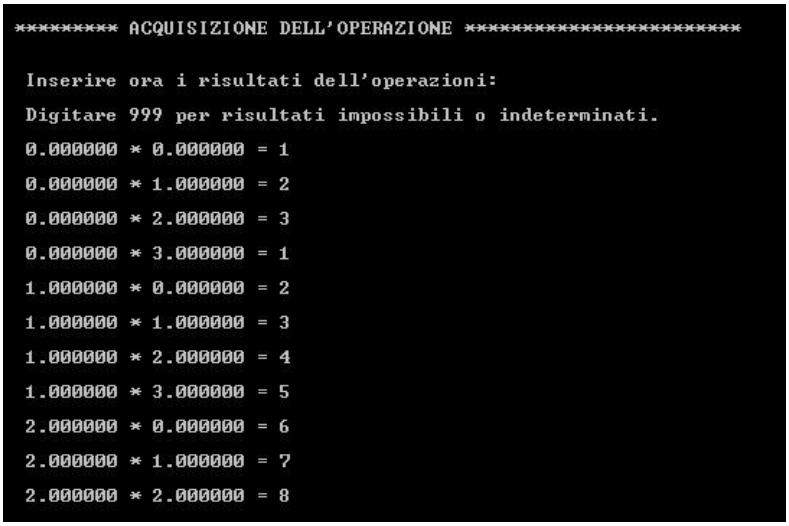


Acquisisco l'insieme e lo stampo per farlo vedere all'utente.

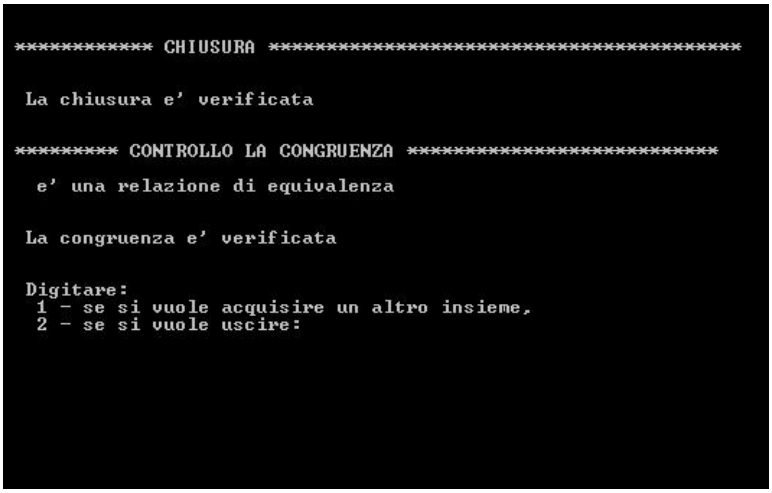


Acquisisco la relazione binaria e la stampo per farla vedere all'utente.

Acquisisco l'operazione acquisendo tutti i risultati possibili.



Inserisco un'operazione chiusa rispetto all'insieme e una relazione che sia una congruenza rispetto l'operazione e verifico l'output.



Inserisco un'operazione non chiusa rispetto all'insieme e verifico l'output.

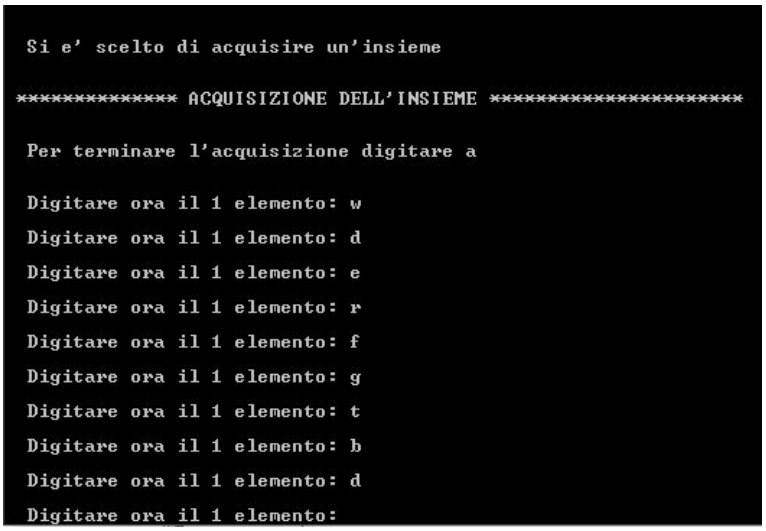
Inserisco un'operazione non chiusa rispetto all'insieme e che la relazione non sia una congruenza rispetto all'operazione e verifico l'output..

```
********** CHIUSURA ****************************************

 La chiusura non e' verificata

********* CONTROLLO LA CONGRUENZA **************************
 Non e'una relazione di equivalenza perche' non e' riflessiva
 Non e'una relazione di equivalenza perche' non e' simmetrica

 La congruenza non e' verificata

 Digitare:
  1 - se si vuole eseguire un altra operazione,
  2 - se si vuole uscire:
```

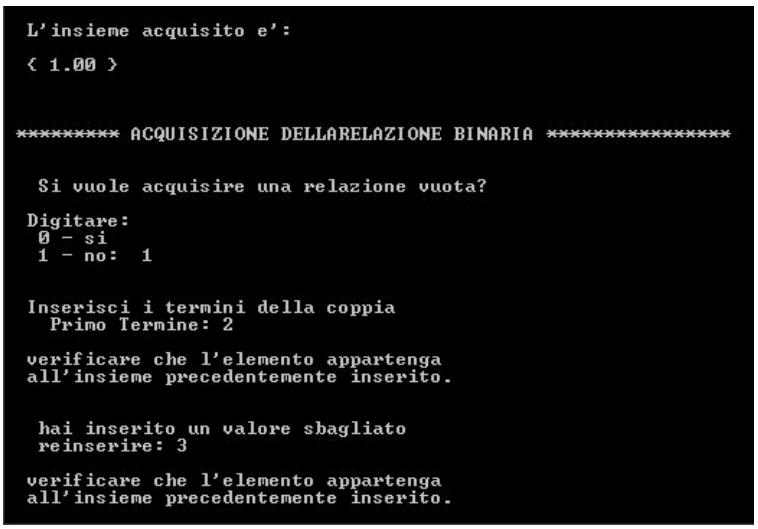
Verifico la presenza di errori durante l'acquisizione della scelta iniziale.

```
*************************************************************
 Questo programma acquisisce nel seguente ordine:

 1) Un insieme;
 2) Una relazione binaria su quell'insieme;
 3) Un'operazione binaria su quell'insieme.

 Poi verifica se l'insieme e' chiuso rispetto all'operazione
 e se la relazione e' una congruenza rispetto all'operazione.

*************************************************************


 Digitare:
  1 - se si vuole iniziare con l'acquisizione dell'insieme,
  2 - se si vuole inserire l'insieme vuoto,
  3 - terminare il programma: a


  hai inserito un valore sbagliato
  reinserire: c


  hai inserito un valore sbagliato
  reinserire: d


  hai inserito un valore sbagliato
  reinserire:
```

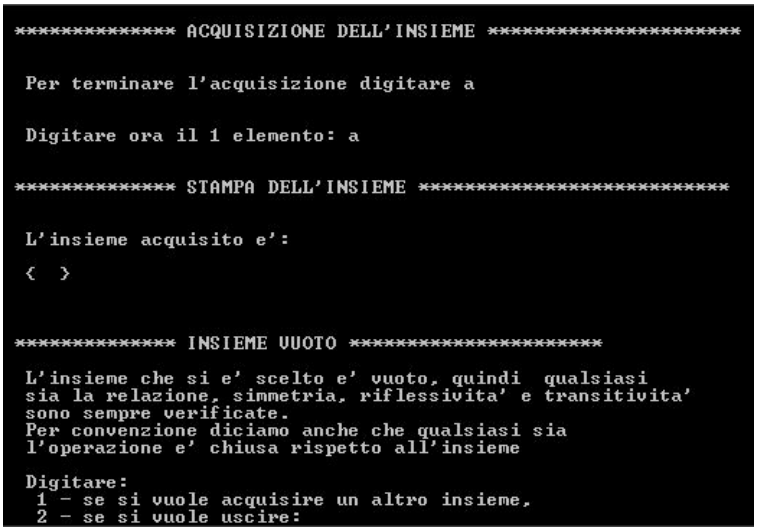
Verifico la presenza di errori durante l'acquisizione dell insieme.



Verifico la presenza di errori durante l'acquisizione della relazione.



Acquisisco l'insieme vuoto.



6 Verifica del programma

Di seguito viene verificata la correttezza di una parte del programma attraverso l'utilizzo di una tripla di Hoare. Si dice tripla di Hoare una tripla nella seguente forma: {Q} S {R}
Dove Q è un predicato detto precondizione, S è un'istruzione ed R è un predicato detto postcondizione. La tripla è vera se e solo se l'esecuzione dell'istruzione S inizia in uno stato della computazione in cui Q è soddisfatta, e termina raggiungendo uno stato della computazione in cui R è soddisfatta.

Istruzione selezionata:
i=0;
if (riscontro != 0)

```
{
   i++;
}
```

sia riscontro $\neq 0 \vee$ riscontro $== 0$
wp(S,R) = $((\beta \rightarrow$ wp(S,R)) $\wedge$ $((\neg\beta$) wp(S,R)))
R => (i = 0) $\vee$ (i = 1)
P => ((riscontro $\neq 0$) $\rightarrow$ ((i = 1) $\vee$ (i = 0))
((riscontro == 0) $\rightarrow$ ((i = 1) $\vee$ (i = 0))
ovvero:
((riscontro $\neq 0$) $\rightarrow$ ((i = 1) $\vee$ (i = 0)) = Vero
((riscontro == 0) $\rightarrow$ ((i = 1) $\vee$ (i = 0)) = Vero
(Vero $\wedge$ Vero) = Vero