Relazione del Progetto relativo l'insegnamento di Programmazione Procedurale per la sessione invernale 2019/2020.

Relatori:
Di Fabrizio Giacomo,
Montanari Matteo Marco

Docente:
Prof. Marco Bernardo

1) Specifica del problema

Scrivere un programma ANSI C che acquisisce da tastiera due matrici di numeri reali allocandole poi dinamicamente e stampa a video la loro somma e il loro prodotto.

2) Analisi del problema

- L’input del problema è costituito da due matrici contenenti numeri reali.

- L’output del problema è costituito dalla somma delle due matrici e dal prodotto della prima matrice per la seconda (il prodotto righe per colonne non è commutativo) nel caso entrambe le operazioni siano legali.

- Le reazioni intercorrenti tra input e output sono le regole della somma e del prodotto righe per colonne tra matrici come definite usualmente in matematica.

3) Progettazione dell'algoritmo

3.1) Scelte di progetto:

- Nel caso in cui non sia possibile eseguire la somma, il prodotto o entrambi si è scelto di comunicare all'utente tale errore e di non richiedere ad esso l'inserimento di un'altra matrice in quanto la soluzione al problema non è sempre garantita.

- Si è scelto di chiedere all'utente all'inizio del programma il numero di righe e di colonne di entrambi le matrici e poi di popolarle entrambe una dopo l'altra poiché più pratico, riducendo così la probabilità di errore.

- Si è scelto, per rappresentare gli input e gli output del problema, l'utilizzo di una struttura dati di tipo array bidimensionale perché particolarmente adeguata.

- Si è scelto di suddividere il programma in funzioni per: la popolazione di una matrice; il calcolo della somma tra due matrici; il calcolo del prodotto tra due matrici; la validazione degli input; la comunicazione della somma e del prodotto. È stato fatto ciò per evitare ridondanza di codice.

3.2) Passi dell'algoritmo:

1. Chiedere numero di righe prima matrice
2. Chiedere numero di colonne prima matrice
3. Chiedere numero di righe seconda matrice
4. Chiedere numero di colonne seconda matrice
5. Chiedere elementi prima matrice
6. Chiedere elementi seconda matrice
7. Calcolare, se possibile, la somma tra prima e seconda matrice
8. Calcolare, se possibile, il prodotto tra prima e seconda matrice
9. Comunicare risultati sotto forma di matrici

4)Implementazione dell'algoritmo:

4.1) Scelte implementative:

- Si è scelto di allocare dinamicamente gli array in questione tramite l'utilizzo di un vettore di puntatori ad array mono-dimensionali di interi e ci riferiremo a questa struttura con l'utilizzo di un puntatore ad interi che punta al primo elemento dell'array di puntatori. Abbiamo effettuato questa scelta perché risulta pratico e comodo allocare una dimensione alla volta.

- Si è scelto inoltre di sfruttare il passaggio di parametri per indirizzo (per ottenere i valori di righe e colonne delle matrici) piuttosto che per valore, in modo da utilizzare meno memoria. Il funzionamento è analogo.

4.2) Makefile e codice sorgente:

Si riportano di seguito il Makefile e il file sorgente:

Makefile:

```
somma_prodotto_matrici: somma_prodotto_matrici.c Makefile
	gcc -ansi -Wall -O somma_prodotto_matrici.c -o somma_prodotto_matrici
pulisci:
	rm -f *.o
pulisci_tutto:
	rm -f operazioni_matrici_dinamiche *.o
```

File sorgente somma_prodotto_matrici.c:

```
/* Progetto d'esame dell'insegnamento di programmazione procedurale relativo il
   corso di laurea in Informatica Applicata L-31 dell'Università di Urbino Carlo
   Bo.

   Autori: Di Fabrizio Giacomo e Montanari Matteo Marco

   Docente: Prof. Marco Bernardo

   Data di fine realizzazione: 01/2020 */


/* Software per il calcolo di somma e prodotto tra matrici allocate dinamicamente:
*/


/* Inclusioni delle librerie standard */


#include <stdio.h>            /* per le operazioni di input / output */

#include <stdlib.h>           /* per l'utilizzo delle funzioni standard di
                                 allocazione e disallocazione di memoria */


/* Dichiarazione delle funzioni */


void valida_matrice(int *,

                    int *,

                    char *);


double **crea_matrice(int *,

                      int *);


void popola_matrice(double **,

                    int *,

                    int *);


double valida_elementi(void);
```

```
double **somma_matrici(double **,
                       double **,
                       int *,
                       int *);

double **prodotto_matrici(double **,
                          double **,
                          int *,
                          int *,
                          int *);

void stampa_matrice(double **,
                    int *,
                    int *);

/* Definizione delle funzioni. */

/* Definizione della funzione main: */

int main (void)
{
      /* Dichiarazioni delle variabili locali alla funzione: */

      /* Lavoro: variabili intere per inizializzazione dei puntatori
         da passare alle funzioni. */

      int inizializzo_punt_1 = 0,   /* variabile il cui indirizzo inizializza il
                                       puntatore righe1 dichiarato successivamente */
          inizializzo_punt_2 = 0,   /* variabile il cui indirizzo inizializza il
```

```
                                puntatore colonne1 dichiarato successivamente*/
        inizializzo_punt_3 = 0,   /* variabile il cui indirizzo inizializza il
                                puntatore righe2 dichiarato successivamente */
        inizializzo_punt_4 = 0;   /* variabile il cui indirizzo inizializza il
                                puntatore colonne2 dichiarato successivamente*/

    /* Input: puntatori da passare alla funzione valida_matrice() per ottenere
       numero di righe e colonne.
       Inizializzazione dei puntatori a locazioni distinte precedentemente
       dichiarate come contenenti variabili intere. */

    int *righe1 = &inizializzo_punt_1,        /*puntatore al valore del numero di
                                              righe della prima matrice */
        *colonne1 = &inizializzo_punt_2,      /*puntatore al valore del numero di
                                              colonne della prima matrice */
        *righe2 = &inizializzo_punt_3,        /*puntatore al valore del numero di
                                              righe della seconda matrice */
        *colonne2 = &inizializzo_punt_4;      /*puntatore al valore del numero di
                                              colonne della seconda matrice */
/* Lavoro: puntatori alle matrici da allocare dinamicamente. */

    double  **mat_1,              /* puntatore alla prima matrice */
            **mat_2,              /* puntatore alla seconda matrice */
            **mat_prodotto,       /* puntatore alla matrice prodotto */
            **mat_somma;          /* puntatore alla matrice somma */
```

```
/* Corpo della funzione main */

/* Validazione dei valori di righe e colonne della prima matrice: */

valida_matrice(righe1,
               colonne1,
               "prima");

/* Validazione dei valori di righe e colonne della seconda matrice: */

valida_matrice(righe2,
               colonne2,
               "seconda");

/* Creazione della prima matrice: */

mat_1 = crea_matrice(righe1,
                     colonne1);



/* Creazione della seconda matrice: */

mat_2 = crea_matrice(righe2,
                     colonne2);
/* Popolazione della prima matrice */

printf("Popolare la prima matrice (A): \n");
```

```
popola_matrice(mat_1,
               righe1,
               colonne1);

/* Popolazione della seconda matrice */

printf("\nPopolare la seconda matrice (B): \n");

popola_matrice(mat_2,
               righe2,
               colonne2);

/* Calcolo della matrice somma, nel caso in cui sia possibile:
   La somma tra due matrici e' possibile solo nel caso in cui le due matrici
   abbiano stesso numero di righe e colonne */

if ((*righe1 == *righe2)&&
    (*colonne1 == *colonne2))
{
      printf("\nLa matrice somma e': \n");

      /* Calcolo della matrice somma */

      mat_somma = somma_matrici(mat_1,
                                mat_2,
                                righe1,
                                colonne1);
```

```
            /* Stampa della matrice somma */

            stampa_matrice(mat_somma,
                           righe1,
                           colonne1);

            /* Disallocazione della memoria riservata alla matrice somma creata
               in seguito alla chiamata a funzione somma_matrici(). */

            free(mat_somma);
      }
      else
            printf("\nImpossibile eseguire somma tra matrici!\n");


      /* Calcolo della matrice prodotto A per B, nel caso in cui sia possibile:
         Il prodotto 'righe-per-colonne' tra matrici e' possibile solo nel caso in
         cui il numero di colonne della prima matrice e' uguale al numero di righe
         della seconda matrice */


if (*colonne1 == *righe2)
      {
            printf("\n La matrice prodotto e':\n");

            /* Calcolo della matrice prodotto */

            mat_prodotto = prodotto_matrici(mat_1,
```

```
                                        mat_2,
                                        righe1,
                                        colonne2,
                                        colonne1);

        /* Stampa della matrice prodotto */

        stampa_matrice(mat_prodotto,
                       righe1,
                       colonne2);

        /* Disallocazione della memoria riservata alla matrice prodotto creata
           in seguito alla chiamata a funzione prodotto_matrici(). */

        free(mat_prodotto);
    }
    else
        printf("\n Impossibile eseguire prodotto tra matrici!\n");


    /* Disallocazione della memoria riservata alle due matrici allocate a
       inizio programma. */

    free(mat_1);
    free(mat_2);

    return (0);

}/* Termine della funzione main. */
```

```
/* Definizione della funzione valida matrice: */


/* La funzione valida_matrice() prende come parametri due puntatori e il nome della
matrice da validare. Questi puntatori puntano a locazioni in cui verranno
memorizzati il numero di righe e di colonne di una matrice dopo averli validati. */


void valida_matrice(int *righe,     /* puntatore al numero di righe della matrice

                                    da validare */

                    int *colonne,   /* puntatore al numero di colonne della
                                    matrice da validare */

                    char  *nome_mat)   /* variabile di tipo stringa contenente il
                                          nome della matrice da validare, il quale
                                          dovrà essere comunicato all'utente */

{

      /* Dichiarazione variabili locali alla funzione: */


      int esito_lettura,      /* lavoro: variabile per conservare il primo valore

                              restituito da scanf() per validazione */

          esito_lettura_2;    /* lavoro: variabile per conservare il secondo
                              valore restituito da scanf() per validazione */


      /* validazione stretta degli input (numero di righe e colonne della matrice

         da validare) */


      do

      {


      /* Acquisizione da tastiera del numero di righe della matrice da validare */


            printf("Inserisci il numero delle righe della %s matrice: \n",

                   nome_mat);
```

```
        esito_lettura = scanf("%d",
                              righe);

    /*Acquisizione da tastiera del numero di colonne della matrice da validare */

        printf("Inserisci il numero delle colonne della %s matrice: \n",
               nome_mat);

        esito_lettura_2 = scanf("%d",
                                colonne);

    /* Validazione degli input digitati */

        if ((esito_lettura != 1) ||
                (esito_lettura_2 != 1) ||
                (*righe <= 0) ||
                (*colonne <= 0))

              printf("Input non accettabile!\n");

        while(getchar() != '\n');

    }while((esito_lettura != 1) ||
           (esito_lettura_2 != 1) ||
           (*righe <= 0) ||
           (*colonne <= 0));
}
```

```
/* Definizione della funzione crea matrice: */


/* La funzione crea_matrice() prende come parametri il numero di righe e colonne
della matrice da creare, e la alloca dinamicamente restituendo un puntatore al
primo elemento della matrice creata. */


double **crea_matrice(int *righe,   /* numero di righe della matrice da creare */
                      int *colonne) /* numero di colonne della matrice da creare */


{
      /* Dichiarazioni variabili locali alla funzione */


      double **mat;     /* output: puntatore al primo elemento della matrice
                           da allocare */
      int i;            /* lavoro: indice per numero di righe della matrice */


      /* Allocazione della prima colonna della matrice in forma di array (di righe
         elementi) di puntatori a valori di tipo double */


      mat = (double **)calloc(*righe,
                              sizeof(double *));



      /* Allocazione di ogni riga della matrice tramite array (di colonne elementi)
         a valori di tipo double */


      for (i = 0;
           (i < *righe);
           i ++)
      {
```

```
            mat[i] = (double *)calloc(*colonne,
                                      sizeof(double));
        }

        return (mat);
}

/* Definizione della funzione popola matrice: */

/* La funzione popola_matrice() prende come parametri la matrice da popolare e il
numero di righe e colonne della matrice stessa.
La funzione chiede in input ogni elemento della matrice da popolare. */

void popola_matrice(double **mat,    /* puntatore al primo elemento della matrice
                                        da popolare */
                    int *righe,      /* numero di righe della matrice da popolare */
                    int *colonne)    /* numero di colonne della matrice da
                                        popolare */

{
        /* Dichiarazione delle variabili locali alla funzione */

        int i,              /* lavoro: indice per numero di righe della matrice */
            j;              /* lavoro: indice per numero di colonne della matrice */

        /* Cicli sulle righe e sulle colonne per popolare ogni elemento
           della matrice */

        for (i = 0;
            (i < *righe);
            i ++)
```

```
        {
                for (j = 0;
                    (j < *colonne);
                    j ++)
                {
                      printf("Digita l'elemento della  %d° riga e %d° colonna: \n",
                                  i+1,
                                  j+1);     /* Aggiustamento della notazione
                                               in quanto gli array nel linguaggio C
                                               partono da 0 */

                      /* Validazione di ogni elemento della matrice tramite la
                         chiamata a valida_elementi() e memorizzazione di tali
                         elementi nella matrice */

                      mat[i][j] = valida_elementi();
                }
        }
}

/* Definizione della funzione valida elementi: */

/* La funzione valida_elementi() chiede in input un elemento e restituisce
   l'elemento validato. La chiamata a valida_elementi() avviene unicamente
   all'interno di popola_matrice() */

double valida_elementi(void)
{
      /* Dichiarazione variabili locali alla funzione: */
```

```
	int esito_lettura;	/* lavoro: variabile per conservare il valore
				   restituito da scanf() per validazione */
	double elemento;	/*input: variabile contenente l'elemento da validare*/

	/* Validazione stretta dei singoli elementi da inserire nella matrice */

	do
	{
		/* Acquisizione da tastiera dell'elemento da validare */

		esito_lettura = scanf("%lf",
					&elemento);

		/* Validazione degli input digitati */

		if (esito_lettura != 1)

			printf("Input non accettabile!\n");

		while(getchar() != '\n');

	}while(esito_lettura != 1);

	return (elemento);
}
/* Definizione della funzione somma tra matrici: */

/* La funzione somma_matrici() prende come parametri le due matrici da sommare e il
```

```
   numero di righe e di colonne di una delle due matrici per creare, calcolare e
   restituisce la matrice somma. Quest'ultima ha come numero di righe e colonne il
   numero di righe e colonne di una delle due matrici indifferentemente */

double **somma_matrici(double **mat_1,     /*  puntatore  al  primo  elemento  della
                                               prima matrice */
                       double **mat_2,     /* puntatore al primo elemento
                                              della seconda matrice */
                       int *righe,         /* numero di righe della prima matrice */
                       int *colonne)       /*numero di colonne della prima matrice*/

{
      /* Dichiarazione delle variabili locali alla funzione: */

      int i,                   /* lavoro: indice per numero di righe della matrice
                                   somma */
          j;                   / *lavoro: indice per numero di colonne della matrice
                                  somma */
      double **mat_somma;      /* output: puntatore al primo elemento della matrice
                                  somma */


      /* Allocazione della matrice somma tramite la chiamata a crea_matrice() */

      mat_somma = crea_matrice(righe,
                               colonne);

      /* Creazione della matrice somma */

      for (i = 0;
          (i < *righe);
```

```
            (i ++))
        {
              for (j = 0;
                   (j < *colonne);
                   (j ++))
              {
                    /* Ogni elemento della matrice somma e' la somma dei rispettivi
                       elementi delle due matrici */

                    mat_somma[i][j] = mat_1[i][j] + mat_2[i][j];
              }
        }

        return (mat_somma);
}

/* Definizione della funzione prodotto tra matrici: */

/* La funzione prodotto_matrici() prende come parametri le due matrici da
moltiplicare, il numero di righe della prima matrice, il numero di colonne della
seconda e il numero di colonne della prima matrice per creare, calcolare e
restituire la matrice prodotto.

Quest'ultima ha come numero di righe il numero di righe della prima matrice e come
numero di colonne il numero di colonne della seconda matrice */

double **prodotto_matrici(double **mat_1, /* puntatore al primo elemento della
                                             prima matrice */
                          double **mat_2, /* puntatore al primo elemento della
                                             seconda matrice */
                          int *righe1,    /* numero di righe della prima
                                             matrice */
                          int *colonne2,  /* numero di colonne della seconda
```

```
                                        matrice */
                        int *colonne1)  /*  numero  di  colonne  della  prima
                                        matrice */

{
    /* Dichiarazione delle variabili locali alla funzione: */

    int i,      /* lavoro: indice per numero di righe della matrice prodotto */
        j,      /* lavoro: indice per numero di colonne della matrice prodotto */
        k;      /* lavoro: indice per il numero di colonne della prima matrice
                   per il calcolo del prodotto */

    double **mat_prodotto;        /* output: puntatore al primo elemento della
                                     matrice prodotto */

    /* Allocazione della matrice prodotto tramite la chiamata a crea_matrice() */

    mat_prodotto = crea_matrice(righe1,
                                colonne2);

    /* Creazione della matrice prodotto */

    for (i=0;
         i < *righe1;
         i++)
    {
       for (j=0;
            j < *colonne2;
            j++)
```

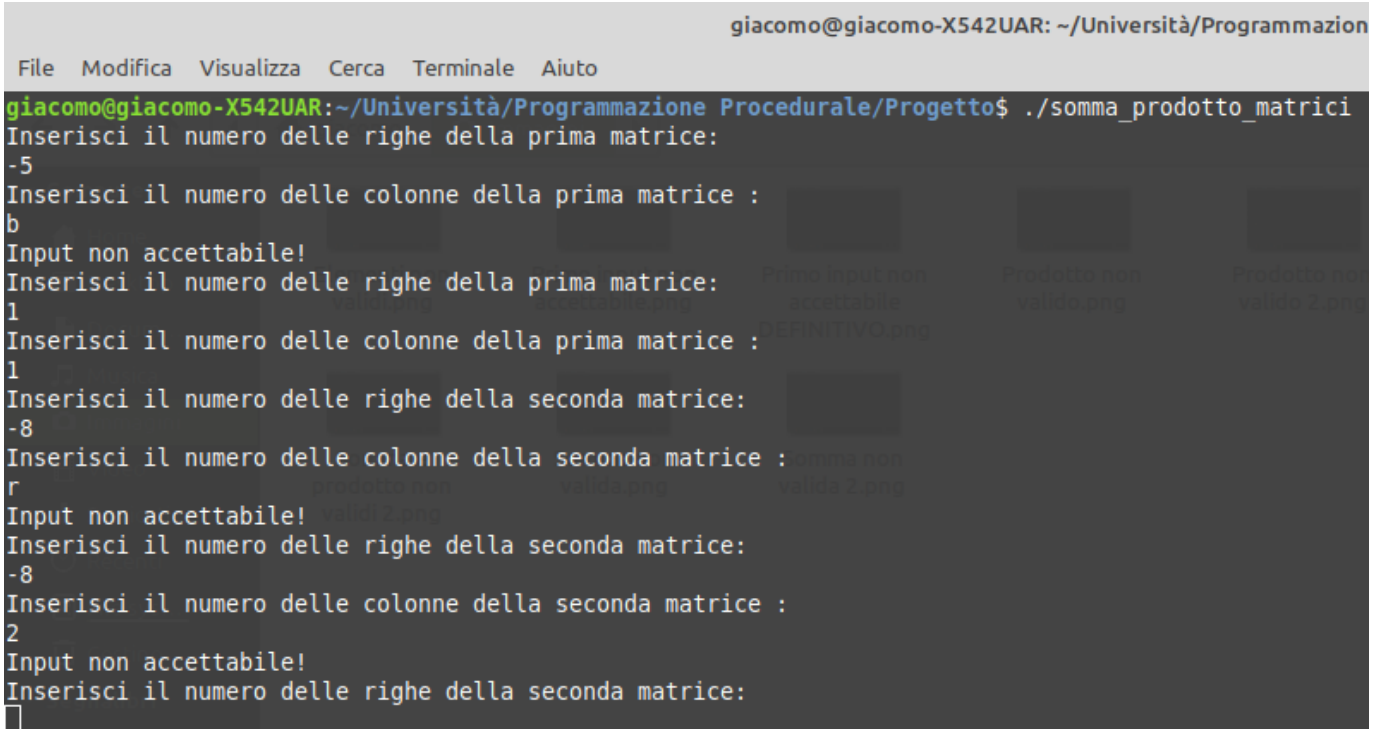
```
        {
            /* Azzeramento dei valori contenuti nella matrice prodotto */

            mat_prodotto[i][j] = 0;

            for (k=0;
                 k < *colonne1;
                 k++)

                 /*prodotto scalare relativo ogni elemento della matrice
                   prodotto*/

                 mat_prodotto[i][j] += mat_1[i][k] * mat_2[k][j];
        }
    }

    return (mat_prodotto);
}

/* Definizione della funzione stampa matrice: */

/* La funzione stampa_matrice() prende come parametri la matrice da stampare e il
suo numero di righe e colonne, e stampa la matrice. */

void stampa_matrice(double **mat,    /* puntatore al primo elemento della matrice da
                                        stampare */
                    int *righe,      /* numero di righe della matrice da stampare */
                    int *colonne)    /* numero di colonne della matrice da
                                        stampare */

{
```

```
        /* Dichiarazione della variabili locali alla funzione: */


        int i,      /* lavoro: indice per numero di righe della matrice
                       da stampare */

           j;       /* lavoro: indice per numero di colonne della matrice
                       da stampare */


        /* Stampa di ogni elemento della matrice */


        for (i = 0;

             (i < *righe);

             (i ++))

        {

              for(j = 0;

                (j < *colonne);

                (j ++))

              {

                   printf("%.2lf",

                          mat[i][j]);


                   printf("\t");

              }


              printf("\n");

         }


        printf("\n");

}

/* Termine del programma. */
```

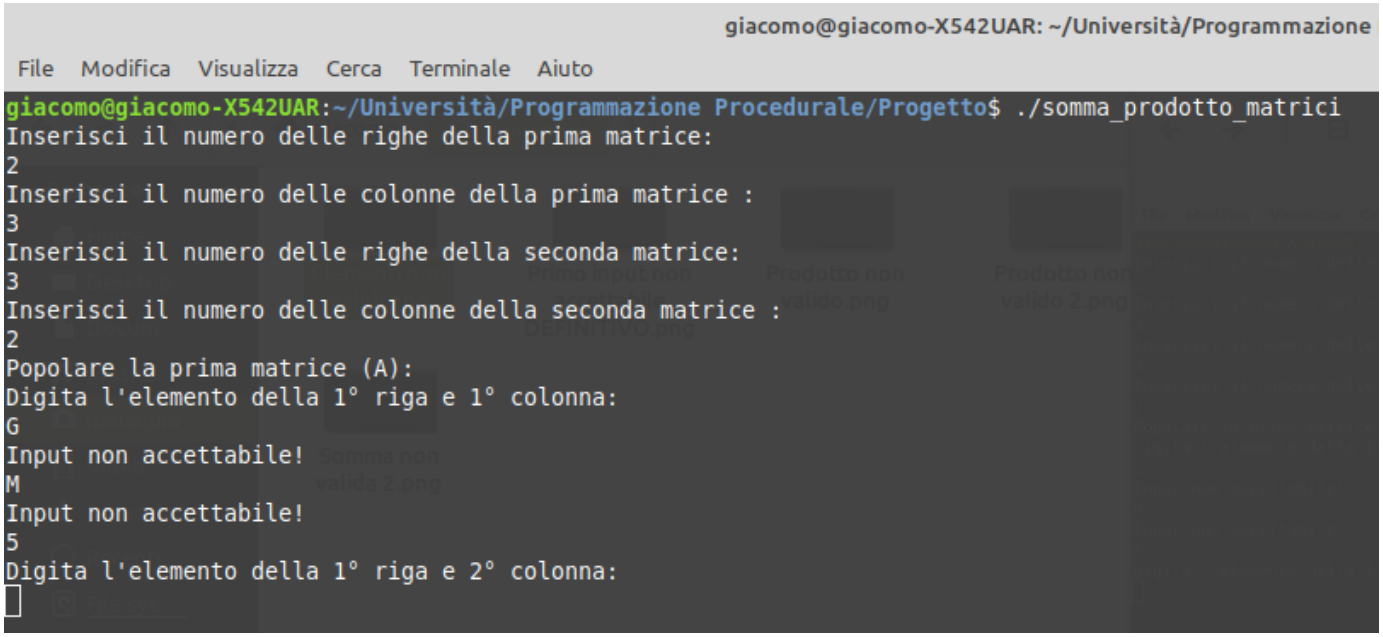
5)Testing del programma

Seguono i test effettuati sul progetto con i vari casi di input.

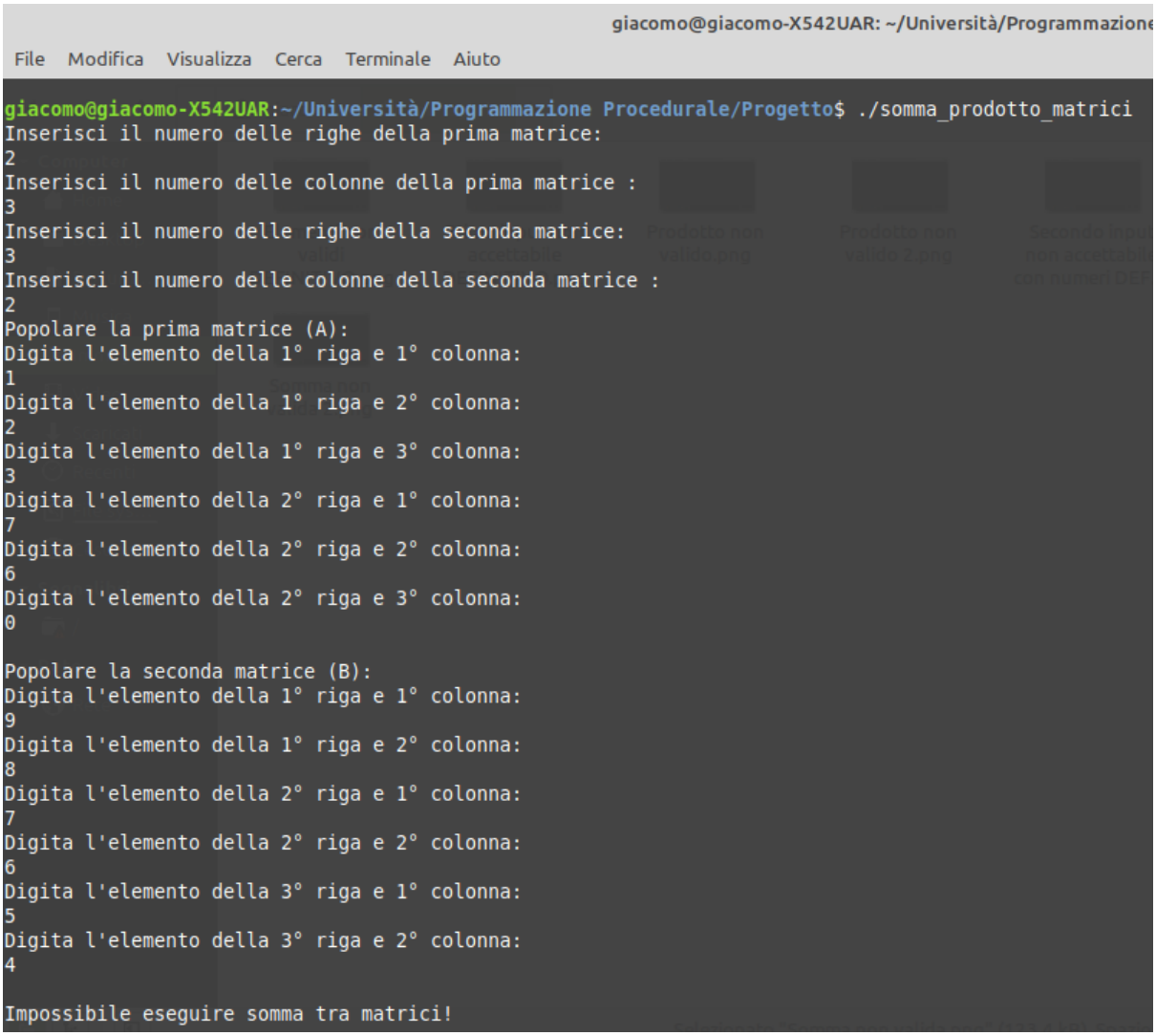
Inserimento di righe o colonne errato (vari casi):

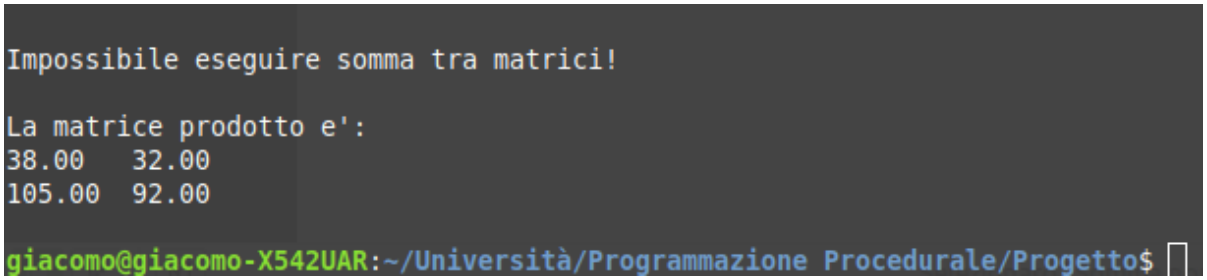


Inserimento di elementi non validi all'interno delle celle della matrice:

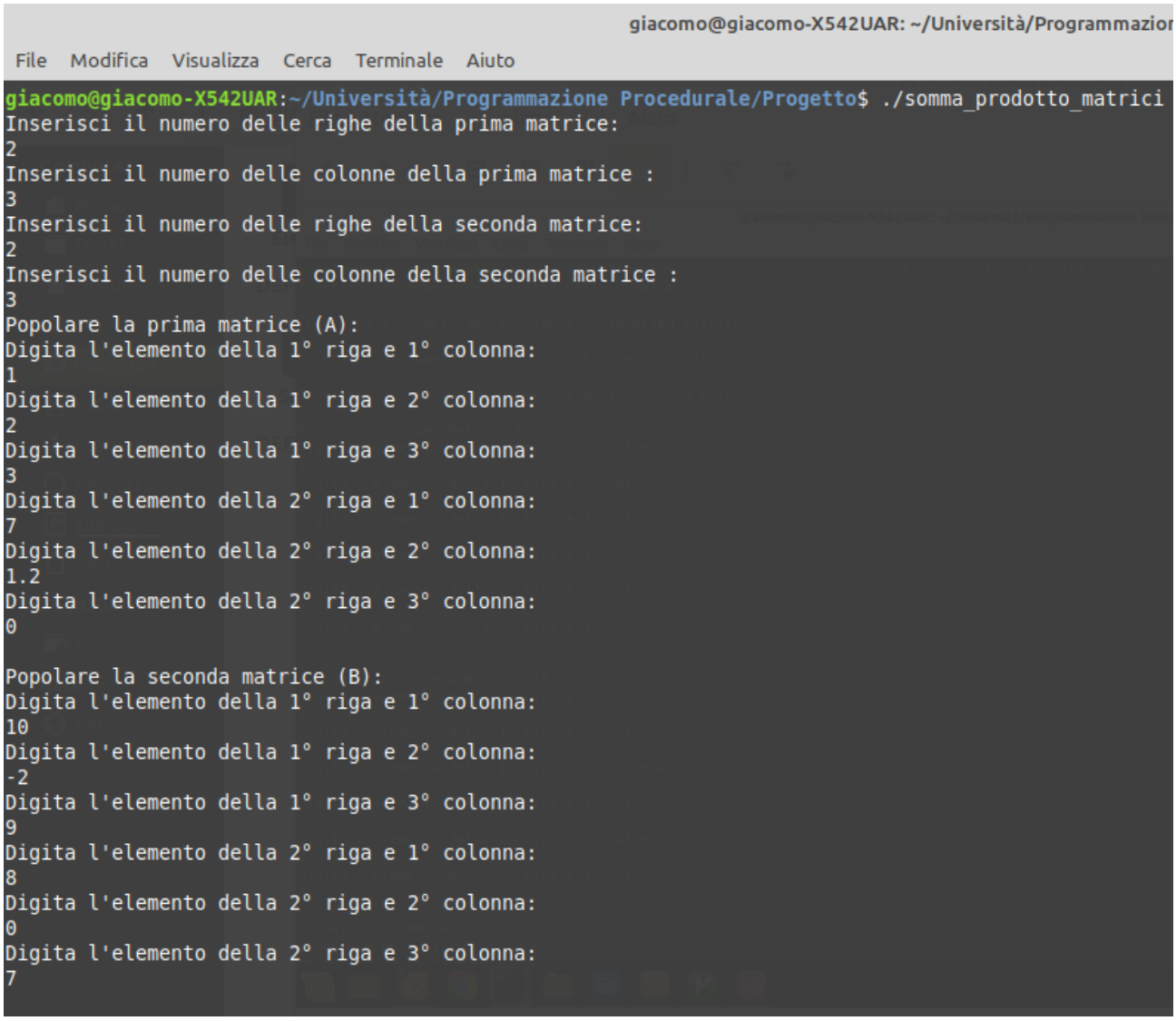


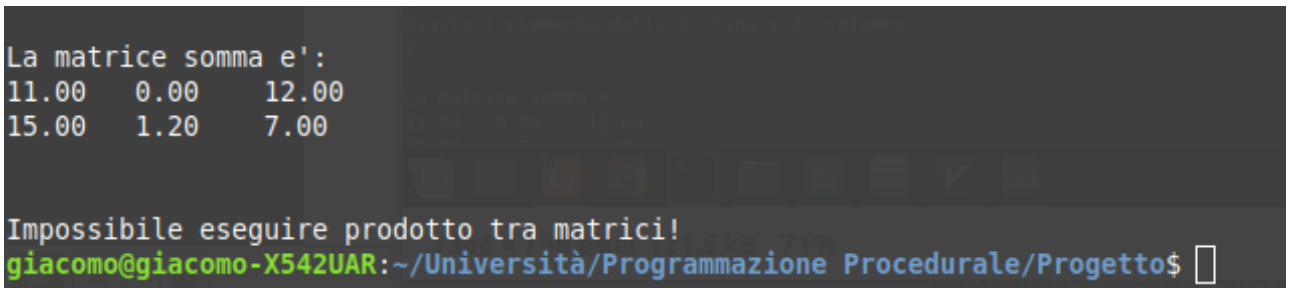
Caso specifico in cui non sia possibile eseguire la somma tra matrici:



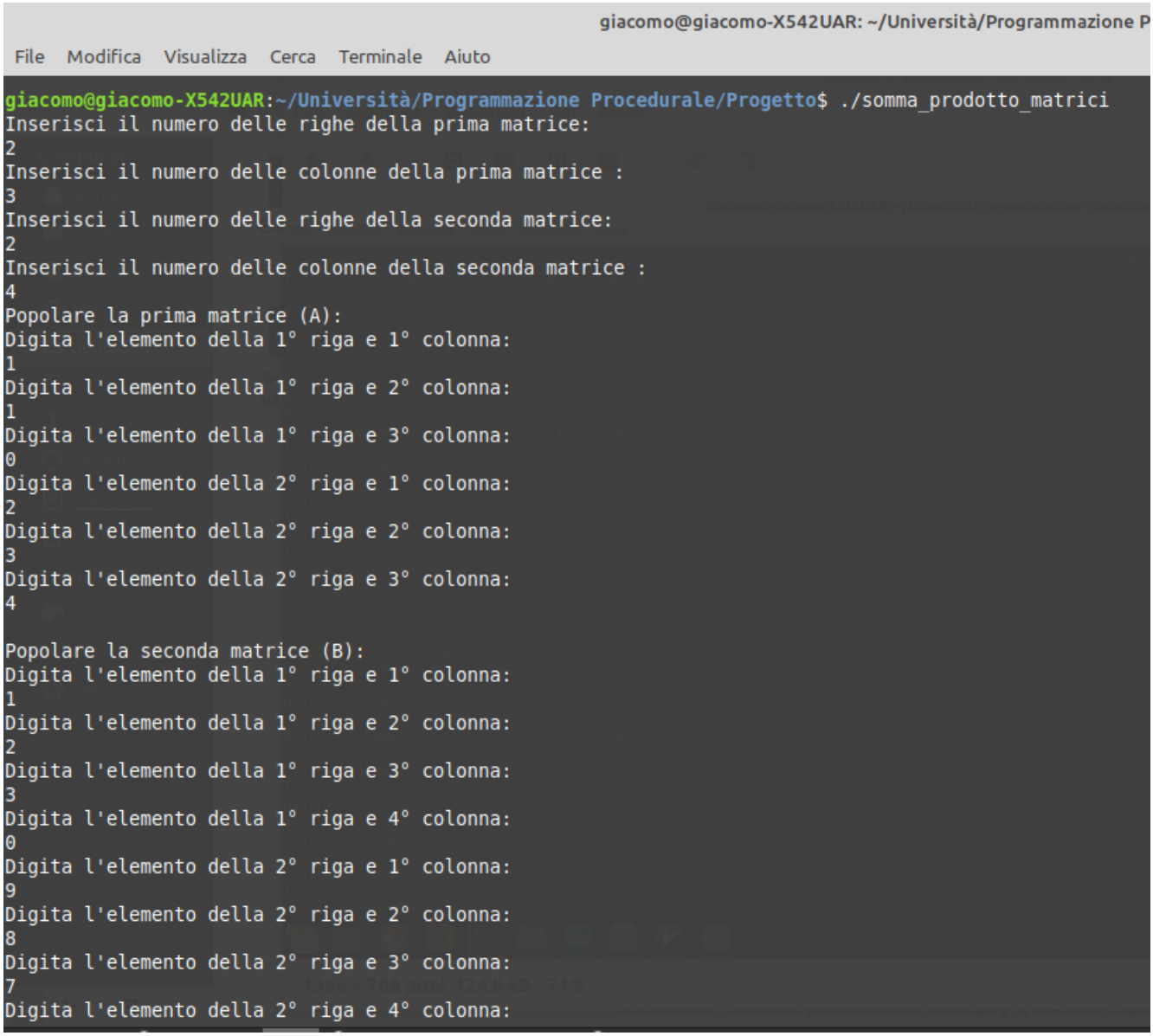


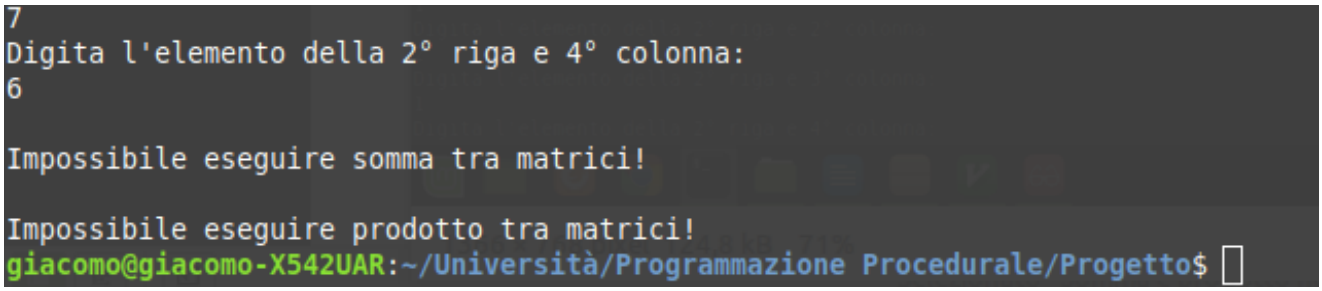
Caso specifico in cui non sia possibile eseguire il prodotto tra matrici:



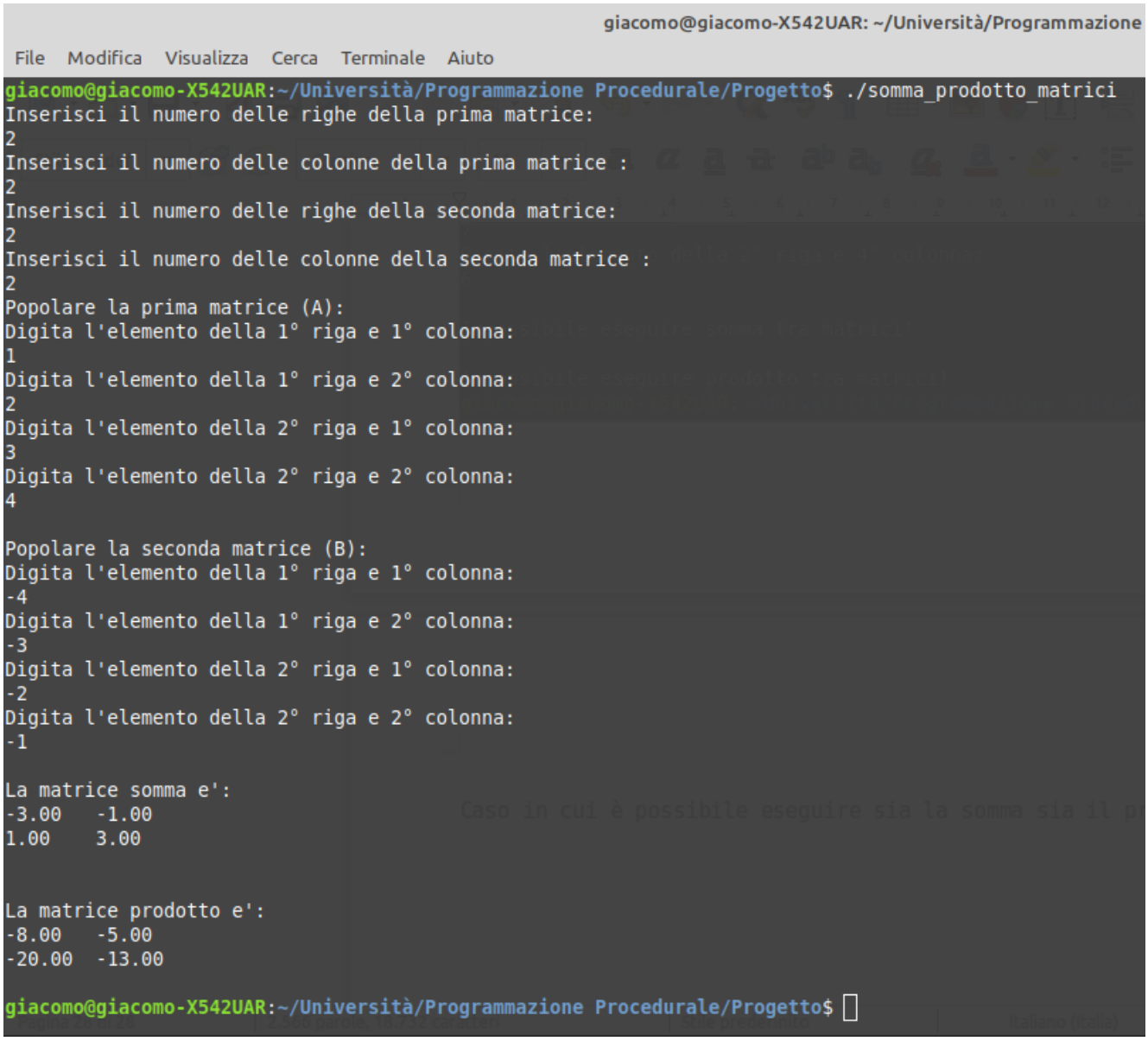


Caso specifico in cui non è possibile né eseguire la somma, né eseguire il prodotto tra matrici:





Caso in cui è possibile eseguire sia la somma sia il prodotto tra matrici:



6)Verifica del programma:

Creazione degli elementi della i-esima riga della matrice somma

6.1) Codice da verificare (dentro la funzione somma_matrici()):

```
for (j = 0;
     (j < *colonne);
     (j ++))
{
       /* Ogni elemento della matrice somma e' la somma dei rispettivi
          elementi delle due matrici */

       mat_somma[i][j] = mat_1[i][j] + mat_2[i][j];
}
```

6.2) Verifica:

A questo punto del programma la variabile intera i contiene un valore fissato e positivo, come anche *colonne e *righe. Poniamo i = I ∈ [0, *righe) e *colonne = C > 0. Il codice da verificare è equivalente al seguente:

```
j = 0;
while (j < C)
        {
                mat_somma[I][j] = mat_1[I][j] + mat_2[I][j];
                j ++ ;
        }
```

Indichiamo l'istruzione di ripetizione " while (β) S " con S′

Allora in base alla post-condizione:

$\{R\} = \{(S_n)_{n=C} = (a_n)_{n=C} + (b_n)_{n=C}\}$,

dove

a_n, b_n, e S_n sono successioni relative a mat_1[I][j], mat_2[I][j] e mat_somma[I][j] rispettivamente.

Per il Teorema dell'invariante di ciclo la tripla {P} S′ {R} è sempre soddisfatta ponendo $\{S_0 = a_0 + b_0\} \equiv$ vero, e verificato che:

$\{P\} = \{0 \leq j < C \wedge S_j = a_j + b_j\}$

$Tr(j) = C - j$

Infatti, sono soddisfatte le seguenti proprietà:

1. *Progresso*
   $\frac{d}{dj} Tr(j) = -1 < 0$

2. *Invarianza* $\{P \wedge \beta\}\, S\, \{P\}$

   $\{P \wedge \beta\} \equiv \{0 \leq j < C \wedge S_j = a_j + b_j \wedge j < C\} \equiv \{0 \leq j < C \wedge S_j = a_j + b_j\}$

   $= \{P\}$

3. *Limitatezza*
   $\{P \wedge Tr(j) \leq 0\} \rightarrow \neg \beta$, $\beta \equiv (j < C)$

   $\{P \wedge Tr(j) \leq 0\} \equiv \{P \wedge C - j \leq 0\} \equiv \{P \wedge j \geq C\} \rightarrow \neg \beta$

Allora l'istruzione

{P} while (β) S {R} ≡ vero

Procediamo infine a ritroso tramite le regole di Dijkstra per ottenere la precondizione dell'intero codice da verificare.

$\{P\}_{j,0} = \{0 \leq 0 < C \wedge S_0 = a_0 + b_0\} \equiv \{ C > 0 \wedge S_0 = a_0 + b_0\} \equiv$ vero

Il codice preso in esame è sempre verificato sotto le ipotesi poste a inizio verifica.

Termine della relazione