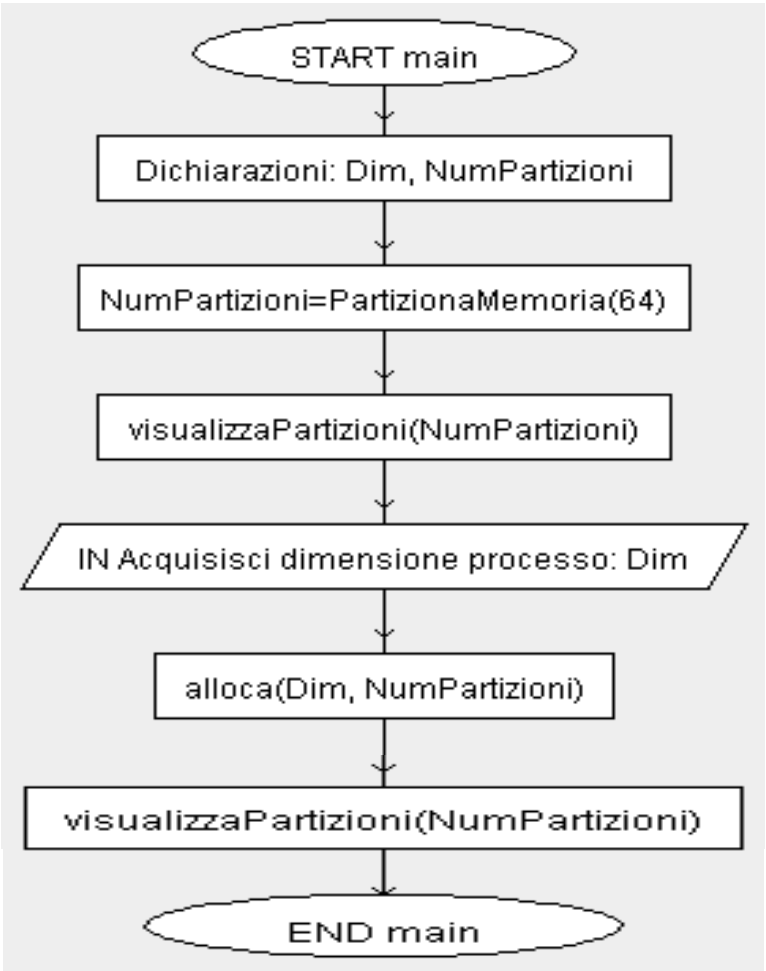
CL4_03_*TECN_PROG* - Esercizi (Gestione della memoria)

1. Scrivere una routine per allocare un processo in una partizione di memoria organizzata a partizioni fisse. La routine usa una tabella delle partizioni con: *indirizzo base*, *dimensione* e *stato* (libera/occupata) di ciascuna partizione. La routine riceve in input l'informazione sulla dimensione del processo, restituisce il numero della partizione scelta (-1 se non è stato possibile allocare il processo) e aggiorna la tabella delle partizioni (lo stato).

2. Scrivere un programma strutturato in sottoprogrammi che, dopo aver suddiviso la memoria secondo la tecnica delle partizioni fisse in un sistema con indirizzi fisici a 16 bit (creazione della tabella delle partizioni generando tutte le partizioni con dimensioni variabili da un minimo di 5 KB a 10 KB ad eccezione eventualmente di una), simuli la richiesta di allocazione in memoria da parte di N processi (tanti quante sono le partizioni) supponendo che ciascuno di essi richieda uno spazio da 1 a 10 KB. Gestire i descrittori dei processi che, per semplicità, si ipotizza siano organizzati in una tabella con i seguenti dati: *PID*, *dimensione della memoria richiesta* in KB, *n° di partizione* in cui è stato allocato il processo (-1 indica che non è stato allocato). Al termine comunicare: l'elenco dei processi allocati, quello dei non allocati, le partizioni occupate con l'eventuale spazio residuo di ciascuna e la percentuale di frammentazione interna definito come rapporto tra la somma delle lunghezze dei frammenti inutilizzati e l'ampiezza complessiva delle partizioni di memoria allocate ai processi.



SimulazionePartizioniFisse.cpp

```
/* Simulazione di organizzazione della memoria secondo la tecnica delle partizioni fisse e allocazione
   della memoria da parte di processi (algoritmo first-fit)*/

#include <iostream>
#include <iomanip>
using namespace std;

#define NumMaxPartizioni 15

// prototipi
   int PartizionaMemoria(int);          // === Suddivide la memoria in partizioni   (IN Dimensione memoria in KB)
                                                                                    (OUT Numero  di partizioni create)
   void visualizzaPartizioni(int);      // === Visualizza la tabella delle partizioni  (IN Numero di partizioni)
   int alloca(int , int);               // === Alloca un processo in memoria        (IN Dimensione di un processo,
                                                                                     IN Numero di partizioni)
                                                                                    (OUT Numero partizione assegnata al
                                                                                    processo / -1 se NON TROVATA)
//struttura
struct partizione
 { int IndBase;
   int Dimensione;
   bool Occupata;    // 0=libera 1=occupata
 };

// variabili globali
// Tabella delle partizioni di memoria
                                          //e inizializzazione
partizione Tabella[NumMaxPartizioni];     //={{122,10,0},{122,50,1},{122,5,0},{122,8,0},{122,12,0}};

//============================================================================
int main( )
{int Dim, NumPartizioni;
   NumPartizioni=PartizionaMemoria(64);

   visualizzaPartizioni(NumPartizioni);

   cout << "Inserisci dimensione del processo: ";
   cin >> Dim;
   cout<<alloca(Dim, NumPartizioni)<<endl;

   visualizzaPartizioni(NumPartizioni);

   system("PAUSE");
   return 0;
}
//============================================================================

// === Alloca un processo in memoria
int alloca(int DimProcesso, int NumPartizioni)
{
  // scorre la tabella delle partizioni per determinare la prima non occupata
  // con dimesione sufficiente a contenere il processo
  for(int np=0; np<NumPartizioni; np++)
      {
       if ( Tabella[np].Dimensione >= DimProcesso  && !Tabella[np].Occupata )
          {
           Tabella[np].Occupata=1;
           return np;
          }
       }
  // non ha trovato nessuna partizione libera di dimensione sufficiente
  return -1;
} //============================================================================
```

```
// === Suddivide la memoria in partizioni

int PartizionaMemoria(int DimMemoria)
{
  int np;                 // numero partizione
  int X;                  // dimensione generata per ogni partizione
  int IndirizzoBase;      // Indirizzo di partenza di ogni partizione
  int DimTotalePartizioni;  // totale progressivo partizioni

  srand(time(NULL));      // inizializza il generatore di numeri casuali

  IndirizzoBase = 0;
  DimTotalePartizioni=0;
  np=0;

  while ( DimMemoria - DimTotalePartizioni > 10)
     {
      X = rand()%6 + 5;          // genero un numero casuale da 5 a 10

      DimTotalePartizioni += X;    // Incremento lo spazio partizionato

      // assegna valori alla partizione
      Tabella[np].IndBase    = IndirizzoBase;
      Tabella[np].Dimensione = X;
      Tabella[np].Occupata   = 0;

      IndirizzoBase = IndirizzoBase + X*1024;
      np++;
     }

 // Controllo se la memoria è stata partizionata completamente
  if (DimTotalePartizioni < DimMemoria)
     {
      // assegna valori all'ultima partizione
      Tabella[np].IndBase    = IndirizzoBase;
      Tabella[np].Dimensione = DimMemoria - DimTotalePartizioni;
      Tabella[np].Occupata   = 0;
      }

  return np+1;
}  //=============================================================================




// === Visualizza la tabella delle partizioni

void visualizzaPartizioni(int NumPartizioni )
{
  cout<<"\n La memoria è suddivisa nelle seguenti Partizioni fisse\n\n";
  cout<<"N.Partiz.    Ind.Base    Dimensione     Occupata \n";
  for (int i=0; i<NumPartizioni; i++)

cout<<setw(5)<<i<<setw(15)<<Tabella[i].IndBase<<setw(10)<<Tabella[i].Dimensione<<setw(15)<<Tabella[i].Occupata<<endl;

}  //=============================================================================
```