```
const int k=3;
const char NOMEFILE[]="file.txt";

struct aereo_t{
   char codice[11];
   bool atterrato;
};

struct aeroporto_t{
   int M;
   aereo_t (*piste)[k];
};

void inizializza_Aeroporto(aeroporto_t &p, int M){

   if(p.M!=0)
      delete[] p.piste;

   p.M=M;
   p.piste = new aereo_t[p.M][k];
   for(int i=0;i<p.M;i++)
      for(int j=0;j<k;j++)
         p.piste[i][j].atterrato=false;

}


void salva_stato_piste(aeroporto_t &p, bool file, ostream
&stream){
   if (file){
      stream<<p.M<<endl;
   }
   for(int i=0;i<p.M;i++){
      stream<<i<<" - ";
      for(int j=0;j<k;j++)
         if(p.piste[i][j].atterrato==false)
            stream<<". ";
         else
            stream<<p.piste[i][j].codice<<" ";
      stream<<endl;
   }
   stream<<endl;
}


bool salva_stato(aeroporto_t &p){


   for(int i=0;i<p.M;i++){;
      for(int j=0;j<k;j++)
         if(p.piste[i][j].atterrato==false)
            f<<". ";
         else
            f<<p.piste[i][j].codice<<" ";
      f<<endl;
   }
   f.close();
   return true;
}

bool carica_stato(aeroporto_t &p){
   ifstream f(NOMEFILE);
   if(!f)
      return false;
   int M;
   f>>M;
   inizializza_Aeroporto(p,M);
   char trattino, indice;
   char tmp[11];

   for(int i=0;i<p.M;i++){
      f>>indice>>trattino;
      for(int j=0;j<k;j++){
         f>>tmp;
         if(strcmp(tmp,".")==0){
            p.piste[i][j].atterrato=false;
            p.piste[i][j].codice[0]='\0';
         }
         else{
            p.piste[i][j].atterrato=true;
            strcpy(p.piste[i][j].codice, tmp);
            }
         }
      }
      f.close();
      return true;
   }



   bool salva_stato(aereoporto_t &p){
      ofstream f(NOMEFILE);
      if(!f)
         return false;
      int tot=p.M*k;
      f.write(reinterpret_cast<const char *>(&p.M), sizeof(int));
      f.write(reinterpret_cast<const char *>(p.porto),
   sizeof(aereoporto_t)*tot);
      f.close();
      return true;
   }

   bool carica_stato(aereoporto_t &p){
      ifstream f(NOMEFILE);
      if(!f)
         return false;

      int M;
      f.read(reinterpret_cast<char *>(&M), sizeof(int));
      inizializza(M, p);
      int tot=p.M*k;

      f.read(reinterpret_cast<char *>(p.porto),
sizeof(aereoporto_t)*tot);

      f.close();
      return true;
   }
```

```
const char nomeFile[] = "lotteria.txt";
const int lunghezzaNome = 15;

struct Biglietto {
    char nome[lunghezzaNome];
    int codice;
};

struct Lotteria {
    int numeroBiglietti;
    int maxDim;
    Biglietto * biglietti;
};

void initLotteria(Lotteria &l){
    l.numeroBiglietti = 0;
    l.maxDim = 1;
    l.biglietti = new Biglietto[l.maxDim];

}

void aumentaBigliettiMax(Lotteria &l){
    int dim = l.maxDim;
    Biglietto * tmpArray = new Biglietto[dim];

    for (int i=0; i<dim; i++)
        tmpArray[i] = l.biglietti[i];

    delete[] l.biglietti;
    l.maxDim *= 2;
    l.biglietti = new Biglietto[l.maxDim];

    for (int i=0; i<dim; i++)
        l.biglietti[i] = tmpArray[i];

    delete[] tmpArray;
}

bool codicePresente(Lotteria &l, int c){
    //trova se il codice è già presente
    for(int i=0; i<l.numeroBiglietti; i++){
        if(l.biglietti[i].codice == c)
            return true;
    }
    return false;
}

bool vendi_biglietto(Lotteria &l, char nome[], int c){
    //controllo sul raggiungimento della dimensione massima
    if(l.numeroBiglietti == l.maxDim)
        aumentaBigliettiMax(l);

    if(!codicePresente(l, c)){
        l.biglietti[l.numeroBiglietti].codice = c;
        strcpy(l.biglietti[l.numeroBiglietti].nome, nome);
        l.numeroBiglietti++;
        return true;
    }
    else
        return false;
}

void scrivi_biglietti(Lotteria &l, ostream &stream, bool
file){
    if(file){
        stream << l.numeroBiglietti << endl;
        stream << l.maxDim << endl;
    }

    for(int i=0; i<l.numeroBiglietti; i++){
        stream << l.biglietti[i].nome << '\t' <<
l.biglietti[i].codice << endl;
    }
}

void carica_biglietti(Lotteria &l){
    ifstream f(nomeFile);
    delete[] l.biglietti;

    f >> l.numeroBiglietti;
    f >> l.maxDim;
    l.biglietti = new Biglietto[l.maxDim];

    for(int i=0; i<l.numeroBiglietti; i++){
        f>>l.biglietti[i].nome;
        f>>l.biglietti[i].codice;
    }
    f.close();
}



int main()
{
    Lotteria lot;
    initLotteria(lot);
    srand(time(0));

      case 1:
           char nome[lunghezzaNome];
           int c;
           cout << "nome: ";
           cin >> nome;
           cout << "codice: ";
           cin >> c;

           vendi_biglietto(lot, nome, c);
           /*
           if(vendi_biglietto(lot, nome, c))
               cout << "venuduto\n";
           else
               cout << "codice già presente\n";
         */
           break;
      case 2:
           scrivi_biglietti(lot, cout, false);
         break;
      case 3:
           {
           ofstream f(nomeFile);
           if(!f)
               cerr << "Errore nell'apertura del file\n";

           scrivi_biglietti(lot, f, true);
           f.close();
           }
         break;
```

```
const int MAX_L = 10;

struct coda{
    int tot_ele = 0, primo = 0, ultimo = 0;
    int v[MAX_L];
};

void inizializza_coda(coda &c){
    for(int i = 0; i < MAX_L; i++)
        c.v[i] = 0;
}

void stampa_coda(coda &c){
    if(c.tot_ele == 0)
        return;
    if(c.primo <= c.ultimo)
        for(int i = c.primo; i <= c.ultimo; i++)
            cout<<c.v[i]<<" ";
    else{
        for(int i = c.primo; i < MAX_L; i++)
            cout<<c.v[i]<<" ";
        for(int i = 0; i <= c.ultimo; i++)
            cout<<c.v[i]<<" ";
    }
    cout<<"Primo: "<<c.primo<<endl;
    cout<<"Ultimo: "<<c.ultimo<<endl;
    cout<<"Elementi totali: "<<c.tot_ele<<endl;

    cout<<endl;
   /*
    for(int i = 0; i < MAX_L; i++){
        cout<<c.v[i]<<" ";
    }
    cout<<endl;
    cout<<"Primo: "<<c.primo<<endl;
    cout<<"Ultimo: "<<c.ultimo<<endl;
    cout<<"Elementi totali: "<<c.tot_ele<<endl;
    */
}

bool inserisci_testa(coda &c, int elem){
    if(c.tot_ele == 0){
        c.v[0] = elem;
        c.tot_ele++;
        return true;
    }
    else if(c.tot_ele != MAX_L){
        if(c.primo == 0)
            c.primo = MAX_L - 1;
        else
            c.primo = c.primo - 1;
        c.v[c.primo] = elem;
        c.tot_ele++;
        return true;
    }
    return false;
}
bool inserisci_coda(coda &c, int elem){
    if(c.tot_ele == 0){
        c.v[0] = elem;
        c.tot_ele++;
        return true;
    }
    else if(c.tot_ele != MAX_L){
        c.ultimo = (c.ultimo+1) % MAX_L;
        c.v[c.ultimo] = elem;
        c.tot_ele++;
        return true;
    }
    return false;
}
int estrai_testa(coda &c){
    if(c.primo == (MAX_L - 1)){
        c.primo = 0;
        c.tot_ele--;
        if(c.tot_ele == 0)
            c.primo = c.ultimo = 0;
        return c.v[MAX_L - 1];
    }
    else if(c.primo == 0 && c.tot_ele == 1){
        c.tot_ele--;
        if(c.tot_ele == 0)
            c.primo = c.ultimo = 0;
        return c.v[c.primo];
    }
    c.tot_ele--;
    c.primo++;
    return c.v[c.primo-1];
}



/*
 Includendo il file di intestazione <ctype>, si posso utilizzare una
 serie di funzioni che operano sui caratteri. Tali funzioni prendono
 in ingresso un valore intero, e ritornano un valore intero.
 Il valore preso in ingresso è interpretato come il codice ASCII di un
 carattere (in ogni caso si puo passare a tali funzioni anche un valore di tipo
 char in ingresso, perchè, come vedremo in seguito, tale valore viene
 convertito implicitamente nel tipo int). A seconda della specifica funzione,
 il valore di ritorno può rappresentare il codice ASCII di un carattere,
 oppure un valore logico (0 per falso, 1 per vero).  Alcune di queste
 funzioni sono:
*/

 // Ritorna un valore diverso da 0 se c e' alfanumerico, 0 altrimenti
 int isalnum(int c) ;

 // Ritorna un valore diverso da 0 se c e' alfabetico, 0 altrimenti
 int isalpha(int c) ;

 // Ritorna un valore diverso da 0 se c e' una cifra decimale, 0 altrimenti
 int isdigit(int c) ;

 // Ritorna un valore diverso da 0 se c e' una cifra esadecimale, 0 altrimenti
 int isxdigit(int c) ;

 // Ritorna un valore diverso da 0 se c e' una lettera minuscola, 0 altrimenti
 int islower(int c) ;

 // Ritorna un valore diverso da 0 se c e' una lettera maiuscola, 0 altrimenti
 int isupper(int c) ;

 // Se c è la codifica di una lettera maiuscola, restituisce la codifica della
 // corrispondente lettera minuscola, altrimenti ritorna c
 int tolower(int c) ;

 // Se c è la codifica di una lettera minuscola, restituisce la codifica della
 // corrispondente lettera maiuscola, altrimenti ritorna c
 int toupper(int c) ;
```

RANDOM

```
srand(time(0));
rand() %(max-min+1)+min;
```

- **strcpy(stringa1, stringa2)**
  copia il contenuto di stringa2 in stringa1 (sovrascrive)
- **strncpy(stringa1, stringa2, n)**
  copia i primi n caratteri di stringa2 in stringa1
- **strcat(stringa1, stringa2)**
  concatena il contenuto di stringa2 a stringa1
- **strcmp(stringa1, stringa2)**
  confronta stringa2 con stringa1: 0 (uguali), >0 (stringa1 è maggiore di stringa 2), <0 (viceversa)

- **write(const char *buffer, int n)**

  Trasferisce i primi *n* byte dell'array buffer sullo stream di uscita.

  Non è aggiunto alcun terminatore.

- LETTURA IN UN BUFFER (2)

read(char *buffer, int n)

Legge *n* byte e li copia nell'array buffer.

Non è previsto alcun delimitatore, né aggiunto alcun terminatore, e ciò la rende una funzione di più basso livello rispetto all'ultimo uso visto della *get*.

- SCRITTURA ARRAY

- scrittura **intera**:

```
void scrivi_array_su_file(const int *a)
{
  ofstream f("file_destinazione") ;
  f.write(
     reinterpret_cast<const char *>(a),
     sizeof(int) * 3
     ) ;
}
```

→ *file_destinazione* sarà un file binario

& non si usa per i vettori