UNIVERSIDAD INCA GARCILASO DE LA VEGA
FACULTAD DE INGENIERIA DE SISTEMAS, COMPUTO Y TELECOMUNICACIONES

ASIGNATURA	Estructura de Información
TEMA	Arreglos
PROFESOR	Carlos A. Ruiz De La Cruz Melo
ALUMNO	
CODIGO	
FECHA	
CICLO	
TURNO	
SEMESTRE	2010-2

1. OBJETIVOS

Que el estudiante:

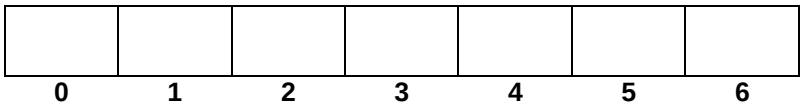
- Asimile los conceptos generales sobre arreglos
- Resolver problemas usando arreglos en una dimensión

2. INTRODUCCION TEORICA

ARREGLOS

Un arreglo (array) es una colección de datos del mismo tipo, que se almacenan en posiciones consecutivas de memoria y reciben un nombre común. Un arreglo es una colección finita, homogénea y ordenada de elementos. **Finita**:Todo arreglo tiene un límite; es decir, debe determinarse cuál será el número máximo de elementos que podrán formar parte del arreglo. **Homogénea**: Todos los elementos del arreglo deben ser del mismo tipo. **Ordenada**: Se puede determinar cuál es el primer elemento, el segundo, el tercero,.... y el n-ésimo elemento.

Conviene imaginar un arreglo como una secuencia contigua de celdas (espacios de memoria), o ***casillas***, en cada una de las cuales se puede guardar un elemento de la colección. Además, es usual dibujarlo como lo ilustra la figura siguiente:



Esta figura representa un arreglo de siete casillas cada una de las cuales se puede utilizar para guardar un dato. La ***dimensión*** o tamaño de un arreglo es el número de casillas que lo conforman. Debe ser claro, entonces, que la figura anterior corresponde a un arreglo de dimensión 7.

Cada una de las casillas de un arreglo tiene asociado un número que la identifica de manera única. A este número se le llama ***índice*** o ***dirección***. En la figura anterior, debajo de cada casilla, aparece su índice.

Para referirse a un determinado elemento de un array se deberá utilizar un índice, que especifique su posición relativa en el array.

Los arreglos se clasifican de acuerdo con el número de dimensiones que tienen. Así se tienen los:

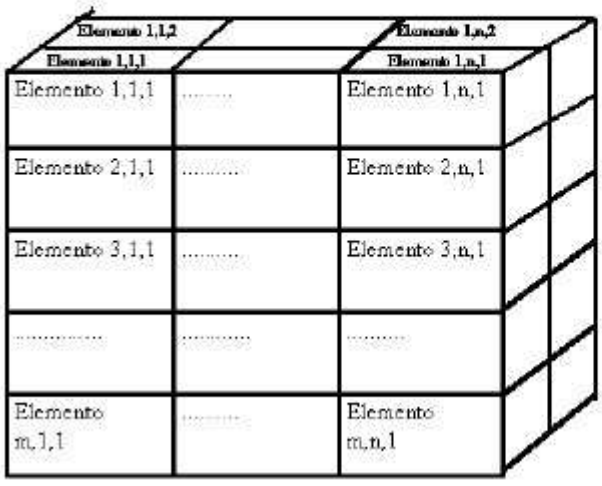
- Arreglos unidimensionales (vectores)
- Arreglos Bidimensionales (tablas o matrices)
- Arreglos Multidimensionales (tres o más dimensiones)

Elemento 1
Elemento 2
Elemento 3
.....
Elemento n

Array Unidimensional

Elemento 1,1	.....	Elemento 1,n
Elemento 2,1	.....	Elemento 2,n
Elemento 3,1	.....	Elemento 3,n
.....	.....	.....
Elemento m,1	.....	Elemento m,n

Array Bidimensional



Array Multidimensional

En C un arreglo se define como:

tipo nombre [*n_elem*];

donde ***tipo*** corresponde al tipo de los elementos que contendrá el arreglo (enteros, reales, caracteres, etc..), ***nombre*** corresponde al nombre o identificador con el cual se denominará el arreglo, y ***n_elem*** corresponde al número de elementos que tendrá el arreglo. A continuación una definición de un arreglo de enteros denominado **entero**:

int **entero**[10];

Para acceder a un elemento del arreglo se utiliza el índice que identifica a cada elemento de manera única. Los índices en C son números enteros correlativos y comienzan desde cero, por lo tanto, si el arreglo contiene **n_elem** elementos el índice del último elemento del arreglo es **n_elem-1**. El siguiente código muestra como se puede inicializar el arreglo del ejemplo anterior, luego de ser declarado:

```
entero[0]=80; //el primer índice es 0
entero[1]=45;
```

```
entero[2]=2;
entero[3]=21;
entero[4]=92;
entero[5]=17;
entero[6]=5;
entero[7]=65;
entero[8]=14;
entero[9]=34; //el último índice del arreglo es 10-1 = 9
```

También se puede declarar e inicializar el arreglo en una sola línea:

```
int arreglo[10]={80, 45, 2, 21, 92, 17, 5, 65, 14, 34};
```

OPERACIONES CON ARREGLOS

Las operaciones que se pueden realizar con arreglos durante el proceso de resolución de un problema son:

- Lectura/ escritura
- Actualización(inserción, eliminación, modificación)
- Recorrido (acceso secuencial)
- Ordenación
- Búsqueda

VENTAJAS Y DESVENTAJAS DE USAR ARREGLOS

VENTAJAS

- Una ventaja que tienen los arreglos es que el costo de acceso de un elemento del arreglo es constante, es decir no hay diferencias de costo entre accesar el primer, el último o cualquier elemento del arreglo, lo cual es muy eficiente.
- Pueden ser usados para implementar otras estructuras de datos sofisticadas como pilas, colas, tablas hash.

DESVENTAJA

- La desventaja es que es necesario definir a priori el tamaño del arreglo, lo cual puede generar mucha pérdida de espacio en memoria si se definen arreglos muy grandes para contener conjuntos pequeños de elementos.
- Su tamaño es fijo, por lo que si no se conoce de antemano el número máximo de elemento a almacenar pueden ocurrir problemas si el espacio reservado es menor del necesario.
- Buscar un elemento en un arreglo desordenado es muy lento
- Insertar elementos de manera ordenada es muy lento

PRIMITIVAS

Para describir las operaciones sobre los arreglos necesitamos hacerlo primero a través de algoritmos, los cuales requieren de primitivas básicas que se presentan a continuación de manera análoga con las primitivas para filas secuenciales.

primitiva	**FILAS: para la fila F**	**ARREGLOS: para el arreglo T con número de elemento N**
Inicio	Inicio(F)	i ←0
Leer	Leer(F, val)	Val ← T[i]
Avanzar	Avanza con cada lectura	i ← i +1
Verificar final	Ultimo (F)	i < N

escribir	Escribir(F, val)	T [i] ← val
terminar	Cerrar(F)	No existe

Ejemplo 1

Escriba un programa que permita registrar y visualizar alumnos en un arreglo. El arreglo debe ser capaz de eliminar un alumno en función de su código.

ESPECIFICACIÓN DEL TAD Y LOS ALGORITMOS

Especificación ALUMNO
variable
entero: codigo
cadena: nombre
operaciones
IngresarAlumno : no retorna valor
MostrarAlumno : no retorna valor
Registrar(a, N, x) : no retorna valor
Visualizar(a , N) : no retorna valor
EliminarPorValor(a , N, cod) : retorna un booleano
significado
En **Registrar** añadir **x** del tipo ALUMNO al arreglo **a** con una cantidad **N** de elementos
En **Visualizar** se muestra el contenido del arreglo **a** de una cantidad **N** de elementos
En **EliminarPorValor** se elimina del arreglo **a** de una cantidad **N** de elementos un alumno de código **cod**.
Fin_ALUMNO

```
funcion EliminarPorValor(a , N, cod): logico
    vector del tipo ALUMNO: a
    entero: N, cod
    Eliminar ←falso
     i ←0
    mientras (i<N)  y ( no Eliminar) hacer
            si (a[i].codigo = cod) entonces
                Eliminar ← verdadero
              sino
                i ←  i + 1
            finsi
    finmientras
    si (EliminarPorValor = verdadero) entonces
       mientras ( i<(N-1)) hacer
             a[i]  ←a[i+1]
             i ←  i +  1
       finmientras
       N ← N - 1
    Finsi
    Retornar Eliminar
 Fin_EliminarPorValor
```

IMPLEMENTACIÓN DEL TAD

```
#include <iostream>
#include <conio.h>
#include "stdio.h"
#include <string>
```

```
using namespace std;

class ALUMNO{
  int codigo;
  string nombre;
 public:
  ALUMNO(){ codigo=0; nombre="";}
  void IngresarAlumno(){
        cout<<"\n leer codigo : ";cin>>codigo;
        cout<<"\n leer nombre : ";cin>>nombre;
     }
  void MostrarAlumno(){
        cout<<"\n "<<codigo<<"      "<<nombre;
  }
  void Registrar(ALUMNO a[],int &N,ALUMNO x ){
     a[N]=x; N++;
  }
  void Visualizar(ALUMNO a[],int N){
      for(int i=0; i<N; i++)
         a[i].MostrarAlumno();
  }
  bool EliminarPorValor(ALUMNO a[],int &N, int cod){
      bool Eliminar=false;
      int i=0;
      while ((i<N) && (!Eliminar)){
        if (a[i].codigo==cod) Eliminar= true;
        else i++;
      }
      if(Eliminar){
          while( i<(N-1)){
             a[i]=a[i+1];
             i++;
          }
          N--;
      }
      return Eliminar;
  }
};

  int main(){
    char op;
    ALUMNO alum[100],a;
    int indice=0,cod;
    for(;;){
       cout<<" \n ingresar     <1>";
       cout<<" \n visualizar   <2>";
       cout<<" \n Eliminar     <3>";
       cout<<" \n salir        <4>";
       op=getch();
       switch(op){
       case '1':a.IngresarAlumno();
                a.Registrar(alum,indice,a);break;
       case '2':a.Visualizar(alum,indice);break;
       case '3':
            cout<<"\n ingrese codigo de alumno a eliminar: ";cin>>cod;
            if(a.EliminarPorValor(alum,indice, cod))
                cout<<"\n elimino alumno";
            else cout<<"\n no elimino alumno";
            break;
       case '4':return 0;
```

```
    }
   }
}
```

Ejemplo 2

Se requiere un programa que permita registrar a los alumnos de la FISC en una fila secuencial, pero no se desea que las operaciones de eliminar por código o inserción en una posición especifica sean lentas así que es necesario que las operaciones se lleven a cabo en memoria, pero sin olvidarse que una vez realizado cualquier numero de operaciones, la información actualizada debe registrarse nuevamente en la fila secuencial.

ESPECIFICACIÓN DEL TAD Y LOS ALGORITMOS

Especificación ALUMNO
variable
entero: codigo
cadena: nombre
operaciones
IngresarAlumno : no retorna valor
MostrarAlumno : no retorna valor
Registrar(a, N, x) : no retorna valor
Visualizar(a , N) : no retorna valor
EliminarPorValor(a , N, cod) : retorna un booleano
InsertarPorPosicion(a, N, x, posición) : retorna un booleano
significado
En **Registrar** añadir **x** del tipo ALUMNO al arreglo **a** con una cantidad **N** de elementos
En **Visualizar** se muestra el contenido del arreglo **a** de una cantidad **N** de elementos
En **EliminarPorValor** se elimina del arreglo **a** de una cantidad **N** de elementos un alumno de código **cod**.
En **InsertarPorPosicion** se inserta en el arreglo **a** de una cantidad **N** de elementos un alumno **x** en una ubicación del arreglo determinado por **posición**
Fin_ALUMNO

```
funcion EliminarPorValor(a , N, cod): logico
    vector del tipo ALUMNO: a
    entero: N, cod
    Eliminar ←falso
     i ←0
    mientras (i<N)  y ( no Eliminar) hacer
            si (a[i].codigo = cod) entonces
                Eliminar ← verdadero
              sino
                i ←  i + 1
            finsi
    finmientras
    si (EliminarPorValor = verdadero) entonces
       mientras ( i<(N-1)) hacer
            a[i]  ←a[i+1]
            i ←  i +  1
       finmientras
       N ← N - 1
    Finsi
    Retornar Eliminar
 Fin_EliminarPorValor

 función InsertarPorPosicion(a , N,  x, pos): lógico
    vector de tipo ALUMNO: a
```

```
ALUMNO : x
entero:pos
Insertar ←falso
 i ←0
mientras (i<N)  y   no InsertarPorPosicion   hacer
        si ( i= pos)
            Insertar ← verdadero
          sino
            i ← i + 1
        finsi
    finmientras
    si InsertarPorPosicion = verdadero entonces
        mientras (i<N) hacer
              temp ←a[i]
              a[i] ←x
              x ←temp
               i ← i + 1
        finmientras
         a[i] ←x
         N ← N +1
    finsi
    retornar Insertar
finInsertarPorPosicion
```

IMPLEMENTACIÓN DEL TAD

```
#include <iostream>
#include <fstream>
#include <conio.h>
#include "stdio.h"

#include <string>
using namespace std;

class ALUMNO{
  int codigo;
  char nombre[40];
 public:
  ALUMNO(){ codigo=0; strcpy(nombre,"");}
  void IngresarAlumno(){
        cout<<"\n leer codigo : ";cin>>codigo;
        cout<<"\n leer nombre : ";cin>>nombre;
    }
  void MostrarAlumno(){
        cout<<"\n "<<codigo<<"      "<<nombre;
  }
  void Registrar(ALUMNO a[],int &N,ALUMNO x ){
     a[N]=x; N++;
  }
  void Visualizar(ALUMNO a[],int N){
     for(int i=0; i<N; i++)
        a[i].MostrarAlumno();
  }
  bool EliminarPorValor(ALUMNO a[],int &N, int cod){
     bool Eliminar=false;
     int i=0;
     while ((i<N) && (!Eliminar)){
       if (a[i].codigo==cod) Eliminar= true;
       else i++;
     }
```

```
    if(Eliminar){
        while( i<(N-1)){
          a[i]=a[i+1];
          i++;
        }
        N--;
    }
    return Eliminar;
}
bool InsertarPorPosicion(ALUMNO a[],int &N, ALUMNO x, int pos){
    bool Insertar=false;
    int i=0;
    ALUMNO temp;
    while ((i<N)&& (!Insertar)) {
        if( i== pos){
           Insertar=true;
        }
        else i++;

    }
    if (Insertar){
       while (i<N){
         temp=a[i];
         a[i]=x;
         x=temp;
         i++;
       }
       a[i]=x;N++;
     }
     return Insertar;
}

void  CargarFila(char fila[],ALUMNO a[],int &N){
      ALUMNO y;
      N=0;
      ifstream lec(fila);                  //(f=fopen(fila,"rb"))==0
      if(!lec){        cout<<"\n No existe aun el archivo "<<fila;getch();     }
      else {       // fread(&a[N],1,sizeof(ALUMNO),f);
        lec.read(reinterpret_cast<char *>(&a[N]),sizeof(ALUMNO));
        while(!lec.eof()){
          N++;
                lec.read(reinterpret_cast<char *>(&a[N]),sizeof(ALUMNO));
        }
               lec.close();   //fclose(f);
      }
   }

void RegistrarFila(char fila[],ALUMNO a[ ],int N){
     ofstream esc(fila, ios::trunk| ios::out| ios::binary);  //(f=fopen(fila,"wb"))==0
     if(!esc){  cout<<"\n ERROR al abrir "<<fila;getch();    }
     else{
       for(int i=0; i<N; i++){ //fwrite(a,N,sizeof(ALUMNO),f);
         esc.write(reinterpret_cast<char *>(&a[i]),sizeof(ALUMNO));
         }
       esc.close();    //fclose(f);
     }
   }
};
```

```
int main(){
  char op;
  ALUMNO alum[100],a;
  int indice,cod, posicion;
  a.CargarFila("filaAA",alum,indice);
  for(;;){
    cout<<" \n ingresar     <1>";
    cout<<" \n visualizar   <2>";
    cout<<" \n Eliminar     <3>";
    cout<<" \n Insertar     <4>";
    cout<<" \n salir        <5>";
    op=getch();
    switch(op){
    case '1':a.IngresarAlumno();
             a.Registrar(alum,indice,a);break;
    case '2':a.Visualizar(alum,indice);break;
    case '3':
        cout<<"\n ingrese codigo de alumno a eliminar: ";cin>>cod;
        if(a.EliminarPorValor(alum,indice, cod))
            cout<<"\n elimino alumno";
        else cout<<"\n no elimino alumno";
        break;
    case '4':cout<<"\n Ingrese posicion de insercion";cin>>posicion;
             a.IngresarAlumno();
             if(a.InsertarPorPosicion(alum,indice,a,posicion))
                 cout<<"\n se inserto correctamente ";
             else cout<<"\n no se inserto el alumno";
             break;
    case '5':a.RegistrarFila("filaAA",alum,indice);return 0;
    }
   }
}
```

3. REQUERIMIENTOS O MATERIAL Y EQUIPO

- Software Dev C++
- 1 Diskete

4. PROCEDIMIENTO

El procedimiento consiste en dos pasos

- Especificación del TAD
- Implementación del TAD

Ejercicio 1

En un archivo de números enteros ha ocurrido un error, el cual consiste en que la información de algunos registros en el archivo principal se ha repetido varias veces existiendo información redundante por lo cual se ha decidido pasar la información sin redundancia a un arreglo.

Especificación del TAD y los algoritmos

Especificación ENTERO
variable
entero: numero
operaciones
IngresarNumero : no retorna valor
RegistrarEnFila(fila, x) : no retorna valor
VisualizarFila(fila) : no retorna valor
VisualizarArreglo(a, N) : no retorna valor
BuscarEnArreglo(dato, a , N) : retorna un booleano
BajarAArreglo(fila, a, N) : no retorna valor
Ecuaciones
En **RegistrarEnFila** añadir **x** del tipo **ENTERO** a la fila secuencial **fila**
En **VisualizarFila** se muestra el contenido de la fila secuencial **fila**
En **VisualizarArreglo** se muestra el contenido del arreglo **a** de **N** elementos
En **BuscarEnArreglo** se busca **dato** del tipo **ENTERO** en el arreglo **a** de **N** elementos.
En **BajarAArreglo** se copia el contenido de la fila secuencial **fila** sin redundancia al arreglo **a** de **N** elementos
Finespecificacion

```
funcion BuscarEnArreglo(dato,  a, N): lógico
  ENTERO: dato
  arreglo del tipo ENTERO: a
  entero: N
  j←0
  BuscarEnArreglo ←falso
  mientras j<n y no Buscar hacer
     si ( dato.numero = v[j].numero ) entonces
          Buscar ← verdadero
          j←j+1
     finsi
  finmientras
  retornar Buscar
fin_BuscarEnArreglo

procedimiento BajarAArreglo (fila, a, N)
   arreglo del tipo ENTERO: a
   entero: N
   inicio(fila)
   leer(fila, dato)
   N←0
   mientras no ultimo(fila) hacer
      si no BuscarEnArreglo(dato, a, N) entonces
           a[N]←dato
           N ←N +1
      finsi
      leer(fila,dato)
    finmientras
    cerrar(fila)
fin_BajarAArreglo
```

Implementación del TAD

```
#include <conio.h>
#include <iostream>
#include <fstream>
using namespace std;

class ENTERO{
  int numero;
  public:
    void IngresarNumero(){
        cout<<"\n leer numero : ";cin>>numero;
    }
    void RegistrarEnFila(char fila[],ENTERO x){
      ofstream esc(fila,ios::app | ios::binary);
      if(!esc){  cout<<"\n ERROR al abrir "<<fila;getch();    }
      else{
         esc.write(reinterpret_cast<char *>(&x),sizeof(ENTERO));
         esc.close();
      }
    }
    void  VisualizarFila(char fila[]){
       ENTERO y;
       ifstream lec(fila);
       if(!lec){        cout<<"\n ERROR al abrir "<<fila;getch();     }
       else {
         cout<<"\n MOSTRANDO DATOS DEL ARCHIVO "<<fila;
         lec.read(reinterpret_cast<char *>(&y),sizeof(ENTERO));
         cout<<"\n";
         while(!lec.eof()){
           cout<<y.numero<<"  ";
                  lec.read(reinterpret_cast<char *>(&y),sizeof(ENTERO));
         }
                lec.close();
       }
    }
    void VisualizarArreglo(ENTERO a[],int N){
        cout<<"\n MOSTRANDO DATOS DEL ARREGLO \n";
        for(int i=0; i<N; i++)
            cout<<a[i].numero<<"  ";
    }

    bool BuscarEnArreglo(ENTERO dato, ENTERO a[], int N ){
         bool EXISTE=false;
         int i=0;
         while ((i < N) && (!EXISTE)){
           if(dato.numero==a[i].numero)
               EXISTE=true;
           i++;
          }
        return EXISTE;
    }
    void BajarAArreglo(char fila[],ENTERO a[], int &N){
       ENTERO y;
       N=0;
       ifstream lec(fila);
       if(!lec){        cout<<"\n ERROR al abrir "<<fila;getch();     }
       else {
         cout<<"\n MOSTRANDO DATOS DEL ARCHIVO "<<fila;
         lec.read(reinterpret_cast<char *>(&y),sizeof(ENTERO));
         while(!lec.eof()){
```

```
          if(!BuscarEnArreglo(y,a,N)){
             a[N]=y;
             N++;
          }
                lec.read(reinterpret_cast<char *>(&y),sizeof(ENTERO));
        }

               lec.close();
      }
    }
};

int main(int argc, char *argv[]){
   char op;
   ENTERO num[100],a;
   int indice=0,cod, posicion;
   for(;;){
      cout<<" \n adicionar  fila1          <1>";
      cout<<" \n Mostrar    fila y arreglo  <2>";
      cout<<" \n Eliminar   redundancia     <3>";
      cout<<" \n salir                      <4>";
      op=getch();
      switch(op){
      case '1':a.IngresarNumero();
               a.RegistrarEnFila("filaP",a);break;
      case '2':a.VisualizarFila("filaP");
               a.VisualizarArreglo(num,indice);
               getch(); break;
      case '3':a.BajarAArreglo("filaP",num,indice);break;
      case '4':return 0;
      }
    }
}
```

Ejercicio 2.

Escriba un programa para un arreglo con igual numero de alumnos aprobados y desaprobados en el cual los alumnos con nota desaprobatoria ocupen las posiciones de los aprobados y viceversa.

Observación

ENTRADA
F1: 11, 20, 05, 18, 09, 10
SALIDA
F2 : 05, 09, 11, 10, 20 ,18

Especificación del TAD y los algoritmos

Especificación ALUMNO
variable
entero: codigo
cadena: nombre
real: promedio
operaciones

IngresarAlumno	: no retorna valor
MostrarAlumno	: no retorna valor
Registrar(a, N, x)	: no retorna valor
Visualizar(a , N)	: no retorna valor
ReubicarArreglo(a , N)	: no retorna valor

significado

En **Registrar** añadir **x** del tipo ALUMNO al arreglo **a** con una cantidad **N** de elementos.
En **Visualizar** se muestra el contenido del arreglo **a** de una cantidad **N** de elementos.
En **ReubicarArreglo** en el arreglo **a** de una cantidad **N** de elementos, las posiciones de los alumnos aprobados se intercambian con los de los desaprobados y viceversa.

Fin_ALUMNO

```
procedimiento ReubicarArreglo(a, N)
   arrreglo del tipo ALUMNO: a
   entero: N
   i←0
   k←0
   mientras (i < N)  hacer
         si (a[i].promedio >= 10.5) entonces
            pos[k]←i
            k ←k + 1
         finisi
         i← i +1
   finmientras
   i←0
   p←0
   mientras (i<N)  hacer
        si(a[i].promedio < 10.5) entonces
              EXISTEPOS←falso
              j←0
              Mientras ( j<k  )  hacer
                    si(pos[j]=i)  entonces
                          EXISTEPOS← verdadero
                    finsi
                    j←j+1
              finmientras
              si( no EXISTEPOS) entonces
                   temp ← a[i]
                   a[i] ← a[ pos[p] ]
                   a[ pos[p] ]← temp
                   p←p+1
              finsi
        finsi
        i←i+1
   finmientras
fin_ReubicarArreglo
```

Implementación del TAD

```
#include <iostream>
#include <conio.h>
#include "stdio.h"
#include <string>
using namespace std;

class ALUMNO{
  int codigo;
  string nombre;
  float promedio;
 public:
  ALUMNO(){ codigo=0; nombre="";}
  void IngresarAlumno(){
       cout<<"\n leer codigo   : ";cin>>codigo;
       cout<<"\n leer nombre   : ";cin>>nombre;
       cout<<"\n leer promedio : ";cin>>promedio;
```

```
   }
 void MostrarAlumno(){
       cout<<"\n "<<codigo<<"      "<<nombre<<"     "<<promedio;
 }
 void Registrar(ALUMNO a[],int &N,ALUMNO x ){
   a[N]=x; N++;
 }
 void Visualizar(ALUMNO a[],int N){
    for(int i=0; i<N; i++)
       a[i].MostrarAlumno();
 }



 void ReubicarArreglo(ALUMNO a[],int N){
   int pos[N];
   int i=0, k=0,p;
   bool EXISTEPOS;
   ALUMNO temp;
   while (i < N){
    if (a[i].promedio >= 10.5){
         pos[k]=i;
         k++;
    }
    i++;
   }
   i=0;p=0;
   while (i<N){
       if(a[i].promedio < 10.5){
          EXISTEPOS=false;
          for(int j=0; j<k;j++)
              if(pos[j]==i) EXISTEPOS=true;
          if(!EXISTEPOS){
              temp = a[i];
              a[i] = a[pos[p]];
              a[pos[p]]= temp;
              p++;
          }
       }
       i++;
   }
  }
};

 int main(int argc, char *argv[]){
   char op;
   ALUMNO alum[100],a;
   int indice=0,cod;
   for(;;){
      cout<<" \n ingresar          <1>";
      cout<<" \n visualizar        <2>";
      cout<<" \n Reordenar arreglo  <3>";
      cout<<" \n salir             <4>";
      op=getch();
      switch(op){
      case '1':a.IngresarAlumno();
               a.Registrar(alum,indice,a);break;
      case '2':a.Visualizar(alum,indice);break;
      case '3':a.ReubicarArreglo(alum,indice);break;
```

```
      case '4':return 0;
      }
    }
}
```

Ejercicio 3.

Se tiene un arreglo de números enteros en el cual existe redundancia de algunos datos, lo que significa que un dato puede encontrarse varias veces. Se pide crear otro arreglo que en cada posición figure un numero del primer arreglo y la cantidad de veces que se repite en el primer arreglo.

Observación

ENTRADA

Arreglo 1 : 1 , 20, 3, 1 , 3, 7, 1, 8

SALIDA

Arreglo 2 1: 3, 20:1, 3:2, 7:1, 8: 1

Ejercicio 4.

En un arreglo de alumnos(código y nombre) existe información redundante de algunos registros, así que se desea eliminar todos los alumnos que tienen redundancia.

Observación

ENTRADA

Arreglo : 11:carlos, 22:maria, 11:carlos, 44:Juana, 11:carlos, 77:Susana, 44:juana

SALIDA

Arreglo : 22:maria, 77:Susana

Ejercicio 5.

Se tiene dos arreglos de alumnos(código y nombre), uno de varones y otro de damas y se desea intercalarlos en otro arreglo.

Observación

ENTRADA

Arreglo 1: 11:carlos, 22: juan

Arreglo 2: 88:maria, 67: elena, 90:angela,66:bertha

SALIDA

Arreglo 3: 11:carlos, 88:maria, 22: Juan, 67: elena, 90:angela,66:bertha

Ejercicio 6

Genere un arreglo de alumnos donde cada alumno tiene como datos codigo, nombre y una nota. Calcule el promedio e indique cuantos alumnos del arreglo son mayores que el promedio y cuantos menores o iguales al promedio.

Ejercicio 7.

Se tienen 3 arreglos de alumnos(código, nombre y nota). Obtenga el promedio de las notas pares y el promedio de las notas impares de todos los alumnos de los tres arreglos y diga cual promedio fue mayor.

5. ANALISIS DE RESULTADOS

- Un arreglo (matriz o vector) es un conjunto finito y ordenado de elementos homogéneos.
- La propiedad “ordenado” significa que el elemento primero, segundo y tercero,…, enésimo de un arreglo puede ser identificado.
- Los elementos del arreglo son homogéneos, es decir, del mismo tipo de datos.
- Un arreglo es una especie de variable que contiene muchos valores pero cada uno con una posición diferente.
- Un arreglo puede ser unidimensional o vectorial, bidimensional o matricial, o multidimencional.

6. BIBLIOGRAFIA

- Estructura de Datos, Un Enfoque Algorítmico. Manuel Gallardo O., Teodomiro Pérez C.
- Estructura de Datos en Pascal, AARON M. TENENBAUM, MOSHE J. AUGENSTEIN
- Fundamentos de Programación, Algoritmos y Estructura de Datos. LUIS JOYANES AGUILAR
- Estructura de Datos y Diseño de Programas. ROBERT L KRUSE