| **TEMA  Nº** | **DETALLE** | **SUBDETALLE** | **CODIGO** | **PALABRAS CLAVE / COMENTARIOS** |
| --- | --- | --- | --- | --- |
| 5 | VECTOR | INICIALIZAR A CERO | void inicializar(int v[],int n) {  for(int i=0;i<n;i++)  v[i]=0; } | Los vectores contienen datos de UNA SOLA CLASE de variable (Por ejemplo, int, unsigned, float, struct, etc) |
| 5 | VECTOR | INGRESAR DATOS | void ingresar(int v[],unsigned t) {  for(unsigned i=0;i<t;i++)  cin>>v[i]; } |  |
| 5 | VECTOR | SUMAR DATOS | int sumar(int v[],int n) {  int sum=0;  for(int j=0;j<n;j++)  sum+=v[j];  return sum; } |  |
| 5 | VECTOR | MOSTRAR VALOR MAYOR | int mayor(int v[],int t) {  int m=0;  for(int i=0;i<t;i++)  {  if(v[i]!=0 && v[i]>m)  m=v[i];  }  return m; } |  |
| 5 | VECTOR | MOSTRAR POSICION DE UN VALOR "X" | void mostrar(int v[],int t,int x) {  for(int i=0;i<t;i++)  {  if(v[i]==x)  cout<<i<<endl;  } |  |
| 5 | VECTOR | MOSTRAR VALOR MAXIMO CON SU POSICION | int mayorConPos(int v[],int t,int vp[],int&j) {  int m=0;  for(int i=0;i<t;i++)  {  if(v[i]!=0 && v[i]>m)  {  m=v[i];  j=0;  vp[j]=i;  j++;  }  else  {  if(m==v[i])  {  vp[j]=i;  j++;  }  }  }  return m; } |  |
| 5 | VECTOR | MOSTRAR TODOS LOS VALORES | void mostrar(int v[],unsigned t) {  for(unsigned i=0;i<t;i++)  cout<<v[i]<<" "<<endl; } |  |
| 5 | VECTOR | MOSTRAR TODOS LOS VALORES  (ACCESO DIRECTO) | void mostrarLeg(int v[],int t) {  for(int i=0;i<t;i++)  {  cout<<v[i]+1<<endl;  } } | En acceso directo se aprovecha el Nº de posición del Vector con algún dato (Por ejemplo Nº de legajo del 0 al 100) |
| 5 | VECTOR | BUSQUEDA SECUENCIAL (1) | int buscSecuencial(int v[],unsigned t,int bus) {  unsigned i=0;  while(i<t && v[i]!=bus)  i++;  if(i==t)  return -1;  else  return i; } | Se recorre todo el vector hasta encontrar el valor buscado |
| 5 | VECTOR | BUSQUEDA SECUENCIAL (2) | int secuencial(int v[],unsigned t,int bus)  {  unsigned i=0;  int pos=-1  while(i<t && pos==-1)  {  if(v[i]==bus)  pos=i;  i++;  }  return pos;  } | Idem anterior. |
| 5 | VECTOR | BUSQUEDA BINARIA | int busqBinaria(int v[],unsigned t,int bus) {  int pos=-1;  unsigned desde=0,hasta=t-1,medio;  while(desde<=hasta && pos==-1)  {  medio=(desde+hasta)/2;  if(v[medio]==bus)  pos=medio;  else  {  if(bus<v[medio])  hasta=medio-1;  else  desde=medio+1;  }  }  return pos; | Solo se utiliza cuando el vector YA ESTA ORDENADO. La búsqueda comienza comparando el valor buscado con el valor de la Posición del Medio del vector.  Si el valor de “Medio” es igual al buscado, entrega esa posición. Si el Valor de “Medio” es menor al buscado, se sigue la búsqueda en la parte “Superior” del vector. Y si no, en la parte “Inferior” del Vector. |
| 5 | VECTOR | ORDENAMIENTO POR BURBUJEO (DO-WHILE) | void burbujeo(int v[],unsigned t) {  unsigned i=0,j;  int aux;  bool cambio;  do  {  cambio=false;  for(j=0;j<t-1-i;j++)  {  if(v[j]>v[j+1])  {  aux=v[j];  v[j]=v[j+1];  v[j+1]=aux;  cambio=true;  }  }  i++;  }while(i<t && cambio); } | Para un orden creciente, se inicia comparando los valores de las posiciones “0” y “1” del vector, intercambiándolos entre si, si estuvieran desordenados.  Luego hace la comparación entre “1” y “2” y procede otra vez al intercambio si fuese necesario, hasta llevarlo a su posición correcta dentro del vector.  Es necesario hacer tantos recorridos como elementos tenga el vector. |
| 5 | VECTOR | ORDENAMIENTO POR BURBUJEO (2 CICLOS FOR) | void burbujeo(int v[], unsigned t)  {  int i , j, aux;  for(i = 0; i <t-1; i++)  {  for(j = 0; j < t-1-i ; j++)  {  if(v[j]>v[j+1])  {  aux=v[j];  v[j]=v[j+1];  v[j+1]=aux;  }  }  }  } | Idem anterior. |
| 5 | VECTOR | APAREO | void listado(Parcial vp1[],int t1,Parcial vp2[],int t2)  {  int i=0,j=0;  while(i<t1 && j<t2)  {  if(vp1[i].nombre==vp2[j].nombre)  {  if(vp1[i].nota>=8 && vp2[j].nota>=8)  {  cout<<vp1[i].nombre<<" Promociona"<<endl;  }  else  {  if(vp1[i].nota>=6 && vp2[j].nota>=6)  cout<<vp1[i].nombre<<" Rinde Final"<<endl;  else  cout<<vp1[i].nombre<<" Recursa"<<endl;  }  i++;  j++;  }  else  {  if(vp1[i].nombre < vp2[j].nombre)  {  cout<<vp1[i].nombre<<" Recursa"<<endl;  i++;  }  else  {  cout<<vp2[j].nombre<<" Recursa"<<endl;  j++;  }  }  }  for(int k=i;k<t1;k++)  cout<<vp1[k].nombre<<" Recursa"<<endl;  for(int k=j;k<t2;k++)  cout<<vp2[k].nombre<<" Recursa"<<endl;  } | **Cada alumno tiene un solo registro en cada uno de los vectores.**  Ejercicio 13) Se tiene un vector de notas (cada elemento tiene nombre del alumno y nota obtenida)  correspondientes al primer parcial y otro vector del mismo tipo del anterior  con las notas correspondientes al segundo parcial,  ambos vectores están ordenados alfabéticamente por nombre.  Se pide hacer un listado ordenado por nombre con la condición obtenida por el alumno |
| 5 | VECTOR | CORTE DE CONTROL | void vectorProm(Parcial vMuchos[],int tM,Parcial vUno[],int &j)  {  int i=0,suma,cont;  j=0;  //string nom;  while(i<tM)  {  vUno[j].nombre=vMuchos[i].nombre;  suma=0;  cont=0;  do  {  suma+=vMuchos[i].nota;  cont++;  i++;  }while(i<tM && vMuchos[i].nombre==vUno[j].nombre);  //vUno[j].nombre=nom;  vUno[j].nota=suma/cont;  j++;  }  } | **Cada alumno tiene más de registro en cada uno de los vectores.**  Ejercicio 15) Se tiene un vector con las notas de los exámenes rendidos por los alumnos  (nombre y nota) ordenado por nombre con un elemento por cada examen rendido  (un alumno puede haber rendido 1, 2, ó más exámenes).  Se pide generar un vector con un elemento por alumno  con el promedio de las notas de sus exámenes, ordenado por nombre. |
| 5 | VECTOR | ORDENAMIENTO POR SELECCIÓN | void seleccion(int v[],unsigned t) {  unsigned i,j;  int menor, pos;  for(i=0;i<t-1;i++)  {  menor=v[i];  pos=i;  for(j=i+1;j<t;j++)  {  if(v[j]<menor)  {  menor=v[j];  pos=j;  }  }  v[pos]=v[i];  v[i]=menor;  } } | Toma el valor primer elemento del Vector y le asigna el valor de “menor”. Luego compara este valor con el resto de los valores. Si encuentra uno menor a él, hace el intercambio, refrescando a “menor”. Luego busca el siguiente elemento menor del vector y lo intercambia con el segundo elemento del vector.  Y en general:  = Busca el mínimo elemento entre una posición i y el final de la lista.  = Intercambia el mínimo con el elemento de la posición i |
| 5 | VECTOR | ORDENAR POR VALOR | void ordenar(Producto v[],unsigned t) {  unsigned i=0,j;  Producto aux;  bool cambio;  do  {  cambio=false;  for(j=0;j<t-1-i;j++)  {  if(v[j].codigo>v[j+1].codigo)  {  aux=v[j];  v[j]=v[j+1];  v[j+1]=aux;  cambio=true;  }  }  i++;  }while(i<t && cambio); } |  |
| 5 | VECTOR | ORDENAMIENTO POR INSERCION (INSERTAR VALOR) | void insercion(int v[],unsigned t) {  unsigned i;  int a,j;  for(i=1;i<t;i++)  {  a=v[i];  j=i-1;  while(j>=0 && v[j]>a)  {  v[j+1]=v[j];  j--;  }  v[j+1]=a;  } } |  |
| 5 | VECTOR | ORDENAR VECTORES PARALELOS | {  unsigned i=1,j;  int aux;  bool cambio;  do  {  cambio=false;  for(j=0;j<t-i;j++)  {  if(vL[j]>vL[j+1])  {  aux=vL[j];  vL[j]=vL[j+1];  vL[j+1]=aux;  aux=vS[j];  vS[j]=vS[j+1];  vS[j+1]=aux;  cambio=true;  }  }  i++;  }while(i<t && cambio); } | Cuando se necesita guardar mas de un dato de una misma entidad (O sea, del mismo elemento), se utilizan vectores en paralelo. Están relacionados entre si por que tienen EL MISMO NUMERO DE POSICION. Cuando se Ordena por unos de los datos, hay que “arrastrar” el orden al resto de los vectores para mantener la coherencia. |
| 5 | VECTOR | MOSTRAR DATOS VECTOR DE STRUCT | struct Producto  {  int codigo;  string descripcion;    };  void mostrarProd(Producto p[],int ce)  {  for(int i=0;i<ce;i++)  cout<<p[i].codigo<<p[i].descripcion<<endl;  } |  |
| 5 | VECTOR | ORDENAR DATOS VECTOR DE STRUCT | struct Producto  {  int codigo;  string descripcion;    };  void ordenar(Producto v[],unsigned t)  {  unsigned i=0,j;  Producto aux;  bool cambio;  do  {  cambio=false;  for(j=0;j<t-1-i;j++)  {  if(v[j].codigo>v[j+1].codigo)  {  aux=v[j];  v[j]=v[j+1];  v[j+1]=aux;  cambio=true;  }  }  i++;  }while(i<t && cambio);  } |  |
| 5 | VECTOR | PASAR VECTOR A ARCHIVO |  |  |
| 5 | VECTOR | PASAR VECTOR A LISTA |  |  |
| 5 | MATRIZ | INICIALIZAR A FALSO | void inic(bool m[][15],int cf, int cc) {  for(int j=0;j<cc;j++)  for(int i=0;i<cf;i++)  m[i][j]=false; } |  |
| 5 | MATRIZ | CARGAR VALORES POR COLUMNA | void cargarPorColum(int matriz[][3],int cf,int cc) {  for(int c=0;c<cc;c++)  {  for(int f=0;f<cf;f++)  cin>>matriz[f][c];  } } |  |
| 5 | MATRIZ | MOSTRAR VALORES POR FILA | void mostrarPorFila(int m[][3],int cf,int cc) {  for(int i=0;i<cf;i++)  {  for(int j=0;j<cc;j++)  cout<<m[i][j]<<" ";  cout<<endl;  } } |  |
| 5 | MATRIZ | SUMAR VALORES DE UNA FILA | void sumatoriaFila(int m[][3],int cf,int cc,int vsf[]) {  for(int j=0;j<cc;j++)  {  for(int i=0;i<cf;i++)  vsf[i]+=m[i][j];  } } |  |
|  | ARCHIVOS |  | \* Buscar y modificar elemento \* Buscar y eliminar Elemento  \* Agregar elemento ordenado |  |
| 7 | ARCHIVOS | MODO APERTURA ARCHIVO BINARIOS |  | rb=lectura, punt al ppio  wb=escritura, punt al ppio  ab=añadir, punt al final  r+b=Lect/Escr, punt al ppio  w+b=Lect/Escr, punto al ppio  a+b=Lect/Añad, punt al final |
| 7 | ARCHIVOS | FUNCIONES ARCHIVO BINARIOS |  | FILE \*f  fopen  fclose  fflush (Vacía contenido)  SEEK\_CUR (Lugar actual)  SEEK\_END (Desde el final)  SEEK\_SET (Desde el inicio)  int ftell (f) Valor actual del indicador  int fseek (f, cant, desde) Define indicador de posición a partir de una posicion |
| 7 | ARCHIVOS | GENERAR, ABRIR Y CARGAR DATOS EN ARCHIVO BINARIO | int main() {  FILE\*archParcial;  char nomArchivo[21];  cout<<"Nombre Archivo (.dat)";  cin>>nomArchivo;  archParcial=fopen(nomArchivo,"wb");  if(archParcial==NULL)  cout<<"ERROR"<<endl;  else  {  Parcial p;  cout<<"DNI ";  cin>>p.dni;  while(p.dni!=0)  {  cout<<"Nombre ";  cin>>p.nom;  cout<<"Nota ";  cin>>p.nota;  fwrite(&p,sizeof(Parcial),1,archParcial);  cout<<"DNI ";  cin>>p.dni;  }  fclose(archParcial);  }  return 0; } | fopen (“archivo”, “wb”)  fwrite  fclose = Cierra el archivo |
| 7 | ARCHIVOS | AGREGAR DATOS A ARCHIVO BINARIO | int main() {  FILE \*aParcial=fopen("Parcial1.dat","ab");  if(aParcial!=NULL)  {  Parcial p;  cout<<"DNI ";  cin>>p.dni;  while(p.dni!=0)  {  cout<<"Nombre ";  cin>>p.nom;  cout<<"Nota ";  cin>>p.nota;  fwrite(&p,sizeof(Parcial),1,aParcial);  cout<<"DNI ";  cin>>p.dni;  }  fclose(aParcial);  } |  |
| 7 | ARCHIVOS | MOSTRAR DATOS DE ARCHIVO | struct Parcial  {  unsigned dni;  char nom[21];  unsigned nota;  };  int main()  {  FILE \*arch;  char nomArchivo[21];  cout<<"Nombre Archivo (.dat) ";  cin>>nomArchivo;  arch=fopen(nomArchivo,"rb");  if(arch==NULL)  cout<<"ERROR"<<endl;  else  {  Parcial par;  fread(&par,sizeof(Parcial),1,arch);  while (!feof(arch))  {  cout<<par.dni<<" "<<par.nom<<" "<<par.nota<<endl;  fread(&par,sizeof(Parcial),1,arch);  }  fclose(arch);  }  return 0;  } |  |
| 7 | ARCHIVOS | CORTE DE CONTROL | struct Parcial  {  unsigned dni;  char nom[21];  unsigned nota;  };  void corteControl(FILE\*a,FILE\*af) {  Parcial p,pf;  unsigned dniA,sum,cont,prom;  char nomA[21];  fread(&p,sizeof(Parcial),1,a);  while(!feof(a))  {  pf.dni=p.dni;  strcpy(pf.nom,p.nom);  sum=0;  cont=0;  do  {  sum+=p.nota;  cont++;  fread(&p,sizeof(Parcial),1,a);  }  while(!feof(a)&&p.dni==pf.dni);  pf.nota=sum/cont;  fwrite(&pf,sizeof(Parcial),1,af);  } } |  |
| 7 | ARCHIVOS | APAREO | struct Parcial  {  unsigned dni;  char nom[21];  unsigned nota;  };  void apareo(FILE\*ap1,FILE\*ap2) {  Parcial p1,p2;  fread(&p1,sizeof(Parcial),1,ap1);  fread(&p2,sizeof(Parcial),1,ap2);  while(!feof(ap1)&&!feof(ap2))  {  if(p1.dni<p2.dni)  {  cout<<p1.dni<<condicion(p1.nota,0)<<endl;  fread(&p1,sizeof(Parcial),1,ap1);  }  else  {  if(p1.dni==p2.dni)  {  cout<<p1.dni<<condicion(p1.nota,p2.nota)<<endl;  fread(&p1,sizeof(Parcial),1,ap1);  fread(&p2,sizeof(Parcial),1,ap2);  }  else  {  cout<<p2.dni<<condicion(0,p2.nota)<<endl;  fread(&p2,sizeof(Parcial),1,ap2);  }  }  }  while(!feof(ap1))  {  cout<<p1.dni<<condicion(p1.nota,0)<<endl;  fread(&p1,sizeof(Parcial),1,ap1);  }  while(!feof(ap2))  {  cout<<p2.dni<<condicion(0,p2.nota)<<endl;  fread(&p2,sizeof(Parcial),1,ap2);  } } |  |
| 7 | ARCHIVOS | GENERAR Y CARGAR DATOS ARCHIVO TEXTO (1) | void armarT() {  FILE\*archivo;  char s[20];  archivo = fopen("texto.txt","w");  if(archivo==NULL)  cout<<"error";  else  {  cin.getline(s,20);  while(strcmp(s,"fin")!=0)  {  fputs(s, archivo);  cin.getline(s,20);  }  fclose(archivo);  } } | fputs (s, f) = Escribe cadena s en flujo |
| 7 | ARCHIVOS | GENERAR Y CARGAR DATOS ARCHIVO TEXTO (2) | void armar() {  FILE \*f;  char nom[31];  int legajo;  int nota;  f = fopen("Alumnos.txt","w");  if(f==NULL)  cout<<"error";  else  {  cout<<"Ingrese legajo (0 para finalizar) ";  cin>>legajo;  while(legajo!=0)  {  cout<<"Nombre: ";  fflush(stdin);  cin.getline(nom,31);  cout<<"Nota: ";  cin>>nota;  fprintf(f,"%d %s %d\n",legajo,nom,nota);  cout<<"Ingrese legajo (0 para finalizar) ";  cin>>legajo;  }  fclose(f);  } } |  |
| 7 | ARCHIVOS | MOSTRAR VALORES ARCHIVO TEXTO (1) | void mostrarT() {  FILE \*archivo;  char c;  archivo = fopen("texto.txt","r");  if(archivo==NULL)  cout<<"error";  else  {  while(!feof(archivo))  {  c=fgetc(archivo);  cout<<c<<endl;  }  fclose(archivo);  } } |  |
| 7 | ARCHIVOS | MOSTRAR VALORES ACHIVO TEXTO (2) | void mostrarT2() {  FILE \*archivo;  char s[20];  archivo = fopen("texto.txt","r");  if(archivo==NULL)  cout<<"error";  else  {  while(!feof(archivo))  {  fgets(s,20,archivo);  cout<<s<<endl;  }  fclose(archivo);  } } |  |
| 7 | ARCHIVOS | BUSQUEDA SECUENCIAL | int busquedaSecuencial(FILE\*arch,unsigned unLeg,Parcial &parc) {  int i=0;  fread(&parc,sizeof(Parcial),1,arch);  while(!feof(arch)&&parc.leg!=unLeg)  {  fread(&parc,sizeof(Parcial),1,arch);  i++;  }   if(feof(arch))  return -1;  else  return i; } |  |
| 7 | ARCHIVOS | BUSQUEDA BINARIA | int busquedaBinaria(FILE\*arch,unsigned unLeg,Parcial &parc) {  unsigned desde, hasta, medio;  desde=0;  fseek(arch,0,SEEK\_END);  //cantidad de registros (Parciales)=ftell(arch)/sizeof(Parcial)  hasta=(ftell(arch)/sizeof(Parcial))-1;  int pos=-1;  while(pos==-1 && desde<=hasta)  {  medio=(hasta+desde)/2;  fseek(arch,medio\*sizeof(Parcial),SEEK\_SET);  fread(&parc,sizeof(Parcial),1,arch);  if(parc.leg==unLeg)  pos=medio;  else  {  if(unLeg<parc.leg)  hasta=medio-1;  else  desde=medio+1;  }  }  return pos; | Cantidad de registros (En este caso = Cantidad de Parciales) de un archivo = ftell(arch)/sizeof(Parcial) |
| 7 | ARCHIVOS | ORDENAR | struct Parcial  {  unsigned dni;  char nom[21];  unsigned nota;  };  void ordenar(Parcial v[],unsigned t)  {  unsigned i=0,j;  Parcial aux;  bool cambio;  do  {  cambio=false;  for(j=0;j<t-1-i;j++)  {  if(v[j].dni>v[j+1].dni)  {  aux=v[j];  v[j]=v[j+1];  v[j+1]=aux;  cambio=true;  }  }  i++;  }while(i<t && cambio);  } |  |
| 7 | ARCHIVOS | ORDENAR POR BURBUJEO | ? |  |
| 7 | ARCHIVOS | ORDENAR POR SELECCIÓN | ? |  |
| 7 | ARCHIVOS | PASAR ARCHIVO A VECTOR |  |  |
| 7 | ARCHIVOS | PASAR ARCHIVO A LISTA |  |  |
| 10 | ESTRUCTURAS DINAMICAS | CREAR NODO | struct Nodo  {  int info;  Nodo\* sgte;  }; |  |
| 10 | ESTRUCTURAS DINAMICAS | CREAR LISTA | struct Alumno  {  unsigned legajo;  char nombre[21];  unsigned nota;  };  void cargarAlumnos(Alumno \*vp[],unsigned c)  {  Alumno \*p;  for(int i=0;i<c;i++)  {  p=new Alumno;  cin>>p->legajo;  cin>>p->nombre;  cin>>p->nota;  vp[i]=p;    }  } |  |
| 10 | ESTRUCTURAS DINAMICAS | CREAR LISTA VECTOR DE PUNTEROS | struct Alumno  {  unsigned legajo;  char nombre[21];  unsigned nota;  };  void cargarAlumnos(Alumno \*vp[],unsigned c)  {  for(int i=0;i<c;i++)  {  vp[i]=new Alumno;  cin>>vp[i]->legajo;  cin>>vp[i]->nombre;  cin>>vp[i]->nota;  }  } |  |
| 10 | ESTRUCTURAS DINAMICAS | BUSCAR NODO EN LISTA | struct Alumno  {  unsigned leg;  char nom[21];  unsigned nota;  };  struct Nodo  {  Alumno info;  Nodo\*sig;  };  Nodo\* busquedaSecM(Nodo\*lista,unsigned unLeg)  {  Nodo\*r=lista;  while(r!=NULL && r->info.leg<unLeg)  r=r->sig;  if(r!=NULL && r->info.leg==unLeg)  return r;  else  return NULL;  } | Devuelve el NODO de un dato buscado (En este caso = unLeg) |
| 10 | ESTRUCTURAS DINAMICAS | INSERTAR ORDENADO EN LISTA | struct Alumno  {  unsigned leg;  char nom[21];  unsigned nota;  };  struct Nodo  {  Alumno info;  Nodo\*sig;  };  void insertar(Nodo\*&lista,Alumno alu)  {  Nodo\*p=new Nodo;  p->info=alu;  Nodo \*ant, \*r=lista;  while(r!=NULL && r->info.leg<alu.leg)  {  ant=r;  r=r->sig;  }  p->sig=r;  if(r!=lista)  ant->sig=p;  else  lista=p;  } |  |
| 10 | ESTRUCTURAS DINAMICAS | BUSCAR E INSERTAR ORDENADO EN LISTA | struct Alumno  {  unsigned leg;  char nom[21];  unsigned nota;  };  struct Nodo  {  Alumno info;  Nodo\*sig;  };  Nodo\*buscarInsertar(Nodo\*&lista,Alumno alu)  {  Nodo \*ant, \*r=lista;  while(r!=NULL && r->info.leg<alu.leg)  {  ant=r;  r=r->sig;  }  if(r!=NULL && r->info.leg==alu.leg)  return r;  else  {  Nodo\*p=new Nodo;  p->info=alu;  p->sig=r;  if(r!=lista)  ant->sig=p;  else  lista=p;  return p;  }  } | Inserta ordenado en lista y muestra la posición donde inserto. |
| 10 | ESTRUCTURAS DINAMICAS | ELIMINAR ELEMENTO DE UNA LISTA | struct Alumno  {  unsigned leg;  char nom[21];  unsigned nota;  };  struct Nodo  {  Alumno info;  Nodo\*sig;  };  void eliminar(Nodo\*&lista,unsigned unLeg)  {  Nodo\*ant,\*r=lista;  while(r!=NULL && r->info.leg<unLeg)  {  ant=r;  r=r->sig;  }  if(r!=NULL && r->info.leg==unLeg)  {  if(r==lista)  lista=r->sig;  else  ant->sig=r->sig;  delete r;  }  else  cout<<unLeg<<" no esta en la lista"<<endl;  } |  |
| 10 | ESTRUCTURAS DINAMICAS | ELIMINAR PRIMERO DE UNA LISTA | struct Alumno  {  unsigned leg;  char nom[21];  unsigned nota;  };  struct Nodo  {  Alumno info;  Nodo\*sig;  };  Alumno sacarPrim(Nodo\*&lista)  {  Alumno a=lista->info;  p=lista;  lista=lista->sig;  delete p;  return a;  } | Elimina primer elemento de la lista, devolviendo el elemento eliminado. |
| 10 | ESTRUCTURAS DINAMICAS | LISTA DOBLE ENLAZADA |  |  |
| 10 | ESTRUCTURAS DINAMICAS | CUENTA CANTIDAD DE NODOS | struct Nodo  {  int info;  Nodo\* sig;  };  int cantidadDeNodos(Nodo\* estructura)  {  int cont = 0;  while(estructura != NULL)  {  cont++;  estructura = estructura->sig;  }  return cont;  } |  |
| 10 | ESTRUCTURAS DINAMICAS | ORDENAR LISTA POR BURBUJEO | void ordenarNumeros(int \*elemento,int nElementos)  {  int aux;  for(int i=0;i<nElementos;i++)  {  for(int j=0;j<nElementos-1;j++)  { if(\*(elemento+j) > (elemento+j+1))  { aux = \*(elemento+j);  \*(elemento+j) = \*(elemento+j+1);  \*(elemento+j+1) = aux;  }  }  }  } |  |
| 10 | ESTRUCTURAS DINAMICAS | MOSTRAR LISTA | struct Alumno  {  unsigned leg;  char nom[21];  unsigned nota;  };  struct Nodo  {  Alumno info;  Nodo\*sig;  };  void listar(Nodo\*lista)  {  Nodo\*r;  r=lista;  while(r!=NULL)  {  cout<<r->info.leg<<r->info.nom<<r->info.nota<<endl;  r=r->sig;  }  } |  |
| 10 | ESTRUCTURAS DINAMICAS | INSERTAR SIN PEDIR MEMORIA | struct Alumno  {  unsigned dni;  string nom;  };  struct Nodo  {  Alumno info;  Nodo\*sig;  };  void insertarSinPedirMemoria(Nodo\*&lista,Nodo\*p)  {  Nodo \*ant, \*r=lista;  while(r!=NULL && r->info.nom<p->info.nom)  {  ant=r;  r=r->sig;  }  p->sig=r;  if(r!=lista)  ant->sig=p;  else  lista=p;  } |  |
| 10 | ESTRUCTURAS DINAMICAS | UBICAR MAXIMO EN LISTA |  |  |
| 10 | ESTRUCTURAS DINAMICAS | UBICAR MINIMO EN LISTA |  |  |
| 10 | ESTRUCTURAS DINAMICAS | UBICAR MAXIMO Y MINIMO EN LISTA | struct Nodo  {  int dato;  Nodo \*siguiente;  };  void calcularMayorMenor(Nodo \*lista)  {  int mayor=0,menor=99999;    while(lista != NULL)  {  if((lista->dato)>mayor)  {  mayor = lista->dato;  }    if((lista->dato)<menor)  {  menor = lista->dato;  }  lista = lista->siguiente;  }    cout<<"\n\nEl mayor numero es:"<<mayor<<endl;    cout<<"El menor numero es: "<<menor<<endl;  } |  |
| 10 | ESTRUCTURAS DINAMICAS | ENCOLAR | struct NodoCola  {  int info;  NodoCola\*sig;  };  void encolar(NodoCola\*&pri,NodoCola\*&ult, int nro)  {  NodoCola\*p=new NodoCola;  p->info=nro;  p->sig=NULL;  if(ult!=NULL)  ult->sig=p;  else  pri=p;  ult=p;  } | Añadir elemento al final de la lista (Cola) |
| 10 | ESTRUCTURAS DINAMICAS | DESENCOLAR | struct NodoCola  {  int info;  NodoCola\*sig;  };  void desencolar(NodoCola\*&pri,NodoCola\*&ult,int &nro)  {  NodoCola\*p=pri;  nro=pri->info;  pri=p->sig;  delete p;  if(pri==NULL)  ult=NULL;  } | Elimina primer elemento de la lista (Cola) FIFO, guardando el elemento eliminado. |
| 10 | ESTRUCTURAS DINAMICAS | APILAR (PUSH) | struct Libro  {  unsigned codigo;  string titulo;  };  struct Nodo  {  Libro info;  Nodo \*sig;  };  void apilar(Nodo \*&pila,Libro lib)  {  Nodo \*p=new Nodo;  p->info=lib;  p->sig=pila;  pila=p;  } | Añadir elemento al inicio de la lista (Pila) |
| 10 | ESTRUCTURAS DINAMICAS | DESAPILAR (POP) | struct Libro  {  unsigned codigo;  string titulo;  };  struct Nodo  {  Libro info;  Nodo \*sig;  };  void desapilar(Nodo\*&pila,Libro&lib)  {  Nodo\*p=pila;  lib=p->info;  pila=p->sig;  delete p;  } | Elimina primer elemento de la lista (Pila) LIFO, guardando el elemento eliminado. |
| 10 | ESTRUCTURAS DINAMICAS | APAREO LISTA | struct Alumno  {  unsigned leg;  string nom;  unsigned nota;  };  struct Nodo  {  Alumno info;  Nodo\*sig;  };  void apareo(Nodo\*lista1, Nodo\*lista2)  {  Nodo \*r1,\*r2;  r1=lista1;  r2=lista2;  while(r1!=NULL && r2!=NULL)  {  if(r1->info.leg<r2->info.leg)  {  cout<<r1->info.leg<<condicion(r1->info.nota,0)<<endl;  r1=r1->sig;  }  else  {  if(r1->info.leg==r2->info.leg)  {  cout<<r1->info.leg<<condicion(r1->info.nota,r2->info.nota)<<endl;  r1=r1->sig;  r2=r2->sig;  }  else  {  cout<<r2->info.leg<<condicion(0,r2->info.leg)<<endl;  r2=r2->sig;  }  }  }  while(r1!=NULL)  {  cout<<r1->info.leg<<condicion(r1->info.nota,0)<<endl;  r1=r1->sig;  }  while(r2!=NULL)  {  cout<<r2->info.leg<<condicion(0,r2->info.leg)<<endl;  r2=r2->sig;  }  } | Dadas la lista del primer parcial y la del segundo parcial mostrar situación |
| 10 | ESTRUCTURAS DINAMICAS | CORTE DE CONTROL LISTA (1 PUNTERO) | struct Alumno  {  unsigned leg;  string nom;  unsigned nota;  };  struct Nodo  {  Alumno info;  Nodo\*sig;  };  void cc(Nodo\*lista)  {  Nodo\*r=lista;  while(r!=NULL)  {  unsigned legA=r->info.leg;  unsigned s=0,c=0;  do  {  s+=r->info.nota;  c++;  r=r->sig;  }while(r!=NULL && r->info.leg==legA);  cout<<legA<<s/c<<endl;  }  } | Dada una lista con alumnos **repetidos** mostrar alumno y promedio |
| 10 | ESTRUCTURAS DINAMICAS | CORTE DE CONTROL LISTA (2 PUNTEROS) | struct Alumno  {  unsigned leg;  string nom;  unsigned nota;  };  struct Nodo  {  Alumno info;  Nodo\*sig;  };  void cc(Nodo\*lista)  {  Nodo\*p=lista;  Nodo\*r=lista;  while(r!=NULL)  {  unsigned s=0,c=0;  do  {  s+=p->info.nota;  c++;  p=p->sig;  }while(p!=NULL && p->info.leg==r->info.leg);  cout<<r->info.leg<<s/c;  r=p;  }  } | Dada una lista con alumnos **repetidos** mostrar alumno y promedio |
| 10 | ESTRUCTURAS DINAMICAS | LISTA CIRCULAR |  |  |
| 10 | ESTRUCTURAS DINAMICAS | LISTA Y SUBLISTA |  |  |
| 10 | ESTRUCTURAS DINAMICAS | PASAR LISTA A VECTOR |  |  |
| 10 | ESTRUCTURAS DINAMICAS | PASAR LISTA A ARCHIVO |  |  |