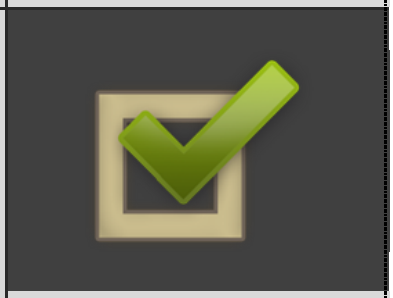
ALGORITMOS Y ESTRUCTURA DE DATOS



Biblioteca de Funciones

Resumen de plantillas Array

Función `agregar`.

Descripción: Agrega el valor `v` al final del *array* `arr` e incrementa su longitud. El control de no superar el tamaño físico del vector es responsabilidad del usuario

```
template <typename T> void agregar(T arr[], int& len, T v)
{
   arr[len]=v;
   len++;
   return;
}
```

Función `buscar`.

Descripción: Busca secuencialmente la primer ocurrencia de `v` en `arr`; retorna su posición o un valor negativo si `arr` no contiene a `v`.

```
template <typename T, typename K>
int buscar(T arr[], int len, K v, int (*criterio)(T,K))
{
   int i=0;
   while( i<len && criterio(arr[i],v)!=0 ){
      i++;
   }
   return i<len?i:-1;
}
```

Función `eliminar`:

Descripción: Elimina el valor ubicado en la posición `pos` del *array* `arr`, decrementando su longitud.

```
template <typename T>
void eliminar(T arr[], int& len, int pos)
{
   int i=0;
   for(int i=pos; i<len-1; i++ ){
      arr[i]=arr[i+1];
   }
   len--;
   return;
}
```

Función `insertar`.

Descripción: Inserta el valor `v` en la posición `pos` del *array* `arr`, incrementando su longitud.

```
template <typename T>
void insertar(T arr[], int& len, T v, int pos)
{
   for(int i=len-1; i>=pos; i--){
      arr[i+1]=arr[i];
   }
   arr[pos]=v;
   len++;
   return;
}
```

Función `insertarOrdenado`.

Descripción: Inserta el valor `v` en el *array* `arr` en la posición que corresponda según el criterio `criterio`.

```
template <typename T>
int insertarOrdenado(T arr[], int& len, T v, int (*criterio)(T,T))
{
   int i=0;
   while( i<len && criterio(arr[i],v)<=0 ){
      i++;
   }

   insertar<T>(arr,len,v,i);

   return i;
}
```

Función `buscaEInserta`.

Descripción: Busca el valor `v` en el *array* `arr`; si lo encuentra entonces retorna su posición y asigna `true` al parámetro `enc`. De lo contrario lo inserta donde corresponda según el criterio `criterio`, asigna `false` al parámetro `enc` y retorna la posición en donde finalmente quedó ubicado el nuevo valor.

```
template <typename T>
int buscaEInserta(T arr[], int& len, T v, bool& enc, int (*criterio)(T,T))
{
   // busco el valor
   int pos = buscar<T,T>(arr,len,v,criterio);

   // determino si lo encontre o no
   enc = pos>=0;

   // si no lo encontre entonces lo inserto ordenado
   if( !enc )
   {
      pos = insertarOrdenado<T>(arr,len,v,criterio);
   }

   return pos;
}
```

Función `ordenar`.

Descripción: Ordena el *array* `arr` según el criterio de precedencia que indica la función `criterio`.

```
template <typename T>
void ordenar(T arr[], int len, int (*criterio)(T,T))
{
   bool ordenado=false;
   int i=1;
   while(!ordenado && i < len - 1)
   {
      ordenado=true;
      for(int j=1; j<len-i; j++){
         if( criterio(arr[j-1],arr[j])>0 ){
            T aux = arr[j-1];
            arr[j-1] = arr[j];
            arr[j] = aux;
            ordenado = false;
         }
      }
   i++;
   }
   return;
}
```

Búsqueda binaria

Función `busquedaBinaria`.

Descripción: Busca el elemento `v` en el *array* `arr` que debe estar ordenado según el criterio `criterio`. Retorna la posición en donde se encuentra el elemento o donde este debería ser insertado.

```
template<typename T, typename K>
int busquedaBinaria(T a[], int len, K v, int (*criterio)(T, K), bool& enc)
{
   int i=0;
   int j=len-1;
   int k=(i+j)/2;
   enc=false;
   while( !enc && i<=j )
   {
      if( criterio(a[k],v)>0 )
      {
         j=k-1;
      }
      else
      {
         if( criterio(a[k],v)<0 )
         {
            i=k+1;
         }
         else
         {
            enc=true;
         }
      }
      k=(i+j)/2;
   }
   return criterio(a[k],v)>=0?k:k+1;
}
template<typename T, typename K>
int busquedaBinaria(T a[], int len, K v, int (*criterio)(T, K), bool& enc)
template<typename T, typename K>
```

```
int busquedaBinaria(T a[], int len, K v, int (*criterio)(T, K), int&p)
//retorna donde lo encontró o -1, en ese caso p es donde debe insertarse para conservar orden
{
   int u=len-1;
   int m, pos = -1;
   while( p <= u && pos == -1)    {
      m = (p+u)/2;
      if( criterio(a[k],v)>0 ){
         j=k-1;
      }
      else{
         if( criterio(a[k],v)<0 ){
            i=k+1;
         }
         else
         {
          pos = m ;
         }
      }
   }
   return pos;
```

```
template<typename T, typename K>
int busquedaBinaria(T a[], int len, K v, int (*criterio)(T, K), int&p)
//con clave multiple retorna el primer elemento que contiene la clave
{
   int u=len-1;
   int m, pos = -1;
   while( p<u)    {
      m = (p+u)/2;
      if( criterio(a[k],v)>0 ){
         j=k-1;
      }
      else{
         if( criterio(a[k],v)<0 ){
            i=k+1;
         }
         else
         {
          pos = m ;
         }
      }
   }
   return pos;
```

```
template<typename T>
int apareo(T a[], T b[], T c[], int n, m, int (*criterio)(T, K), int&p)
//n y m son los topes de los vectores a y b respectivamente
{
   int i= 0;
   int j= 0;
   int k= 0;

   while( i<n || j<m){

      if( criterio(a[i],v[j]>0 )
      {
         c[k] = a[i];
         i++;
      }
      Else
      {
         c[k] = b[j];
         j++;
      }
      k++;
   }
   return pos;
}
```

Resumen de plantillas Archivos

Template: read

```
template <typename T> T read(FILE* f)
{
   T buff;
   fread(&buff,sizeof(T),1,f);
   return buff;
}
```

Template: readN

```
template <typename T> void read(FILE* f, T buff[], int N)
{
   T buff;
   fread(buff,sizeof(T),N,f);
   return;
}
```

Template: write

```
template <typename T> void write(FILE* f, T v)
{
   fwrite(&v,sizeof(T),1,f);
   return;
}
```

Template: writeN

```
template <typename T> void write(FILE* f, T v[],int N)
{
   fwrite(v,sizeof(T),N,f);
   return;
}
```

Template: seek

```
template <typename T> void seek(FILE* arch, int n)
{
   // SEEK_SET indica que la posicion n es absoluta respecto del inicio del archivo
   fseek(arch, n*sizeof(T),SEEK_SET);
}
```

Template: fileSize

```
template <typename T> long fileSize(FILE* f)
{
    // tomo la posicion actual
   long curr=ftell(f);
  // muevo el puntero al final del archivo
  fseek(f,0,SEEK_END);   // SEEK_END hace referencia al final del archivo
   // tomo la posicion actual (ubicado al final)
  long ultimo=ftell(f);
   // vuelvo a donde estaba al principio
  fseek(f,curr,SEEK_SET);
  return ultimo/sizeof(T);
}
```

Template: filePos

```
template <typename T> long filePos(FILE* arch)
{
   return ftell(arch)/sizeof(T);
}
```

Template: busquedaBinaria

```
template <typename T, typename K>
int busquedaBinaria(FILE* f, K v, int (*criterio)(T,K))
{
    // indice que apunta al primer registro
   int i = 0;
   // indice que apunta al ultimo registro
   int j = fileSize<T>(f)-1;
   int k = (i+j)/2;

   seek<T>(f,k);
   // leo el registro que se ubica en el medio, entre i y j
   T r = leerArchivo<T>(f);
   while( i<=j && criterio(r,v)!=0 ){
       // si lo que encontre es mayor que lo que busco...
      if( criterio(r,v)>0 ){
         j = k-1;
      }
      Else {
         // si lo que encontre es menor que lo que busco...
         if( criterio(r,v)<0 ) {
            i=k+1;
         }
      }
      // vuelvo a calcular el indice promedio entre i y j
      k = (i+j)/2;

      // posiciono y leo el registro indicado por k
      seek<T>(f,k);

      // leo el registro que se ubica en la posicion k
      r = leerArchivo<T>(f);
   }

   // si no se cruzaron los indices => encontre lo que busco en la posicion k
   return i<=j?k:-1;
}
```

Ejemplos

Leer un archivo de registros usando el *template* `read`.

```
f = fopen("PERSONAS.DAT","r+b");
// leo el primer registro
Persona p = read<Persona>(f);
while( !feof(f) )
{
   cout << p.dni<<", "<<p.nombre<<", "<<p.altura << endl;
   p = read<Persona>(f);
}
fclose(f);
```

Escribir registros en un archivo usando el *template* `write`.

```
f = fopen("PERSONAS.DAT","w+b");
// armo el registro
Persona p;
p.dni = 10;
strcpy(p.nombre,"Juan");
p.altura = 1.70;
// escribo el registro
write<Persona>(f,p);
fclose(f);
```

Acceso directo a los registros de un archivo usando los templates `fileSize`, `seek` y `read`.

```
f = fopen("PERSONAS.DAT","r+b");
// cantidad de registros del archivo
long cant = fileSize<Persona>(f);
for(int i=cant-1; i>=0; i--)
{
   // acceso directo al i-esimo registro del archivo
   seek<Persona>(f,i);
   Persona p = read<Persona>(f);
   cout << p.dni<<", "<<r.nombre<<", "<< r.altura << endl;
}
fclose(f);
```