Funciones para estructuras enlazadas

Sin Plantilla

El Nodo

```
struct Nodo1 {
int  info;
Nodo* sig;
};
Nodo1* p1 =
new Nodo1();
```

```
struct Nodo2 {
string  info;
Nodo* sig;
};
Nodo2* p2 =
new Nodo2();
```

```
struct Alumno{
  int Legajo;
  string Nombre;
}
```

```
void  push1 (Nodo1 * &p1, int x);
void  push2 (Nodo2 * &p2, string x);
void  push3 (Nodo3 * &p3, Alumno  x);
```

Invocación – Ejemplo de uso

```
int * p1 = NULL;
push1(p1,1);
string* p2 = NULL;
push2(p2,"Juan");
Alumno * p3 = NULL;
push3(p3,crear(123,"juan"));
```

Requiere una función diferente para cada tipo de dato.

Con plantilla para un dato genérico en info

El Nodo

```
template <typename T>
struct Nodo {
T info;
Nodo<T>* sig;
};
Nodo <T>* P = new Nodo<T>();
```

```
template <typename T>
void push(Nodo<T>* & p, T v)
```

Invocacion – Ejemplo de uso

```
Nodo<int>* p1 = NULL;
agregarAlFinal<int>(p1,1);
Nodo<string>* p2 = NULL;
agregarAlFinal<string>(p2,"Juan");
Nodo<Alumno>* p3 = NULL;
agregarAlFinal<Alumno>(p1,crear(123, "juan"))
```

Utiliza la misma función para tipos diferentes.

Funciones para pilas

```
void push (Nodo* &p, int v){
Nodo* nuevo = new Nodo();
nuevo->info = v;
nuevo->sig = p;
p = nuevo;
}
// push(pila,4) solo para enteros
int pop (Nodo* &p)
{
Nodo* aux = p;
int v = aux->info;
p = aux->sig;
delete aux;
return v;
}

//x = pop(pila) x es entero
```

```
template <typename T>
void push(Nodo<T>* & p, T v){
Nodo<T>* nuevo = new Nodo<T>();
nuevo->info = v;
nuevo->sig = p;
p = nuevo;
}
// push<tipo>(pila, x) para tipo generico
template <typename T>
T pop(Nodo<T>* & p)
{
Nodo<T>* aux = p;
T v = aux->info;
P = aux->sig;
delete aux;
return v;
}
// x=pop<tipo>(pila) x es generico
```

Funciones para colas

```
void agregarr (Nodo* &p, Nodo*& q,  int v)
{
  Nodo* nuevo = new Nodo();
  nuevo->info = v;
  nuevo->sig = NULL;
  if (p == NULL)
     p = nuevo;
  else
     q-> sig = Nuevo;
  q = Nuevo;
}

int suprimirr (Nodo* &p, Nodo* &q, int v)
{
    Nodo* aux = p;
    int v = aux->info;
    p = aux->sig;
    if (p==NULL) q = NULL;
    delete aux;
return v;
}
```

```
template <typename T>
void agregar (Nodo<T>* &p, Nodo<T>*& q, T v)
{
  Nodo<T>* nuevo = new Nodo<T>();
  nuevo->info = v;
  nuevo->sig = NULL;
  if (p == NULL)
     p = nuevo;
  else
     q-> sig = Nuevo;
  q = Nuevo;
}

T suprimirr (Nodo<T>* &p, Nodo<T>* &q, T v)
{
   Nodo<T>* aux = p;
   T v = aux->info;
    p = aux->sig;
    if (p==NULL) q = NULL;
    delete aux;
return v;
}
```

Funciones para listas Con criterio para implementar distintas formas de ordenamiento

```
Nodo* insertarOrdenado(Nodo*& l, int v )
{
   //crear el nodo
   Nodo* nuevo = new Nodo();
   nuevo->info = v;
   nuevo->sig = NULL;
  //si esta vacia o va delante del primero
   if (l==NULL || v < l->info )
      l = nuevo;
      return nuevo;
  //si no salió, no esta vacía y va entre dos o al final
     Nodo* aux = l;
   while( aux->sig !=NULL &&
          v > aux->sig->info)  {
         aux = aux->sig;
    }
// enlaza los punteros
    nuevo -> sig = aux sig,
    aux -> sig = nuevo;
   return nuevo;
}//fin insertar ordenado
```

```
template <typename T>
Nodo<T>* insertarOrdenado(Nodo<T>*& l, T v,
int (*criterio)(T,T) ){
   //crear el nodo
   Nodo<T>* nuevo = new Nodo<T>();
   nuevo->info = v;
   nuevo->sig = NULL;
  //si esta vacia o va delante del primero
   if (l==NULL || criterio(l->info,v)>0 )
      l = nuevo;
      return Nuevo;
  //si no salió, no esta vacía y va entre dos o al final
     Nodo<T>* aux = l;
   while( aux->sig !=NULL &&
            criterio(aux->sig->info,v)<0   {
       aux = aux->sig;
    }
// enlaza los punteros
    nuevo -> sig = aux sig,
    aux -> sig = nuevo;
   return nuevo;
}//fin insertar ordenado
//criterio(lo del nodo, lo nuevo) si es  > 0 lo pone adelante; < 0 sigue buscando
```

```
Nodo* buscarEnLista(Nodo *l, int  v ){
    Nodo* q = l;
    while( q !=NULL && q->info 1= v)   {
       q = q->sig;
    return q;
 }


// Retorna la dirección de memoria que contiene el valor
buscado o NULL en caso de no encontrarlo
```

```
template <typename T>
Nodo<T>* buscarEnLista(Nodo<T>*l, T v, int
(*criterio)(T,T) ){
    Nodo<T>* q = l;
    while( q !=NULL &&
           criterio(aux->sig->info,v)!=0   ){
       q = q->sig;
       return q;
   }
// Retorna la dirección de memoria que contiene el valor
buscado o NULL en caso de no encontrarlo


buscarOinsertarOrdenado(Nodo<T>*& l, T v ){
   //inserta sin repetir la clave, el modelo mas simple es
buscar esa clave y solo insertar si no la encuentra
```

```
//Un ejemplo completo con listas enlazadas utilizando templates
#include <iostream>
using namespace std;
template <typename T>
struct Nodo {
  T info;      // valor que contiene el nodo
  Nodo<T>* sig; // puntero al siguiente nodo
};

template <typename T>
void liberar(Nodo<T>*& l)
{
  Nodo<T>* aux=NULL;
  while( l!=NULL )
  {
    aux = l;
   aux = aux->sig;
   delete l;
  }
}// fin liberar la lista

template <typename T>
Nodo<T>* insertarOrdenado(Nodo<T>*& l, T v, int (*criterio)(T,T) )
{
  Nodo<T>* nuevo = new Nodo<T>();
  nuevo->info = v;
  nuevo->sig = NULL;

  Nodo<T>* aux = l;
  Nodo<T>* ant = NULL;
  while( aux!=NULL && criterio(aux->info,v)<=0 )  {
    ant = aux;
    aux = aux->sig;
  }

  if( ant==NULL )  {
    l = nuevo;
  }
  else  {
    ant->sig = nuevo;
  }
  nuevo->sig = aux;
  return nuevo;
}//fin inserter ordenado otro modelo distinto al desarrollado mas arriba
```

```
struct Alumno { //declaración de la estructura a utilizar
  int leg;
  string nom;
};
Alumno crearAlumno(int leg, string nom) //función auxiliar para crear un registro
{
  Alumno a;
  a.leg = leg;
  a.nom = nom;
  return a;
}
int criLeg(Alumno a1, Alumno a2)
{//establece el criterio de orden retorna negative, cero o positivo segun los valores analizados
  return a1.leg-a2.leg;
}
int criNom(Alumno a1, Alumno a2)
{
  return a1.nom<a2.nom?-1:a1.nom>a2.nom?1:0;
}
int main()// programa principal
{
  Alumno a;
  Nodo<Alumno>* p = NULL;
  // inserto ordenado x legajo
  a = crearAlumno(30,"Juan");
  insertarOrdenado<Alumno>(p,a,criLeg);
  // muestro el contenido de la lista
  Nodo<Alumno>* aux = p;
  while( aux!=NULL )  {
    cout << aux->info.leg << ", " << aux->info.nom << endl;
    aux=aux->sig;
  }
  liberar<Alumno>(p);
  // ahora una lista ordenada por nombre
  p = NULL;
  a = crearAlumno(30,"Juan");
  insertarOrdenado<Alumno>(p,a,criNom);
  // muestro el contenido de la lista
  aux = p;
  while( aux!=NULL )  {
    cout << aux->info.leg << ", " << aux->info.nom << endl;
    aux=aux->sig;
  }
  liberar<Alumno>(p);
  return 0;
}
```