Algoritmos y Estructura de Datos. K1041. 2do Examen Parcial.
Fecha: 27/10/2017
Apellido y nombre: …… **Legajo:** ……..

Para aprobar debe sumar como mínimo 60 puntos (calif. 6), siendo 80 puntos el mínimo requerido para aprobación directa (calificación 8).

1) Algoritmo. Seleccione la respuesta correcta e ingrese un comentario describiendo su decisión.

```
template <typename T> Nodo<T>* miFuncion(Nodo<T>* la,
Nodo<T>* lb, Nodo<T>* lc) {
        while(la!=NULL&&lb!=NULL) {
                if(la->info<=lb->info) {
                        agregar(lc,la->info);
                        la=la->sig;
                } else {
                        agregar(lc,lb->info);
                        lb=lb->sig;
                }
        }
        while(la!=NULL) {
                agregar(lc,la->info);
                la = la->sig;
        }
        while(lb!=NULL) {
                agregar(lc,lb->info);
                lb = lb->sig;
        }
        return lc;
}
```

¿Qué resultado produce esta función?

[ ] Inserta nodos ordenados por el campo info

[ ] Intercala los nodos de dos listas

[ ] Retorna el nodo que coincide con T

[ ] Ninguna de las 3 anteriores

Comentario/supuestos sobre la respuesta seleccionada:

……

……

……

(20 puntos)

2) Chat. Un sistema de atención al público necesita poner en espera las solicitudes de conversación con los representantes. Como la demanda supera a la cantidad de operadores, el objetivo es que los clientes queden en espera manteniendo el orden de llegada. Cada cliente es representado en el sistema con los siguientes datos:

id_cliente	**nickname**
entero	cadena de caracteres

Se pide: Crear un programa que agregue clientes a una estructura de datos desde la función ***agregarCliente()*** a medida que se van registrando en el sistema.
También implementar la función ***atenderProximo()*** que retorne los datos del siguiente cliente a atender. Crear las estructuras necesarias.

(30 puntos)

3) Videojuego. Se debe crear una función que cargue en memoria el historial de partidos de fútbol disputado por cada equipo desde un archivo binario de registros. El videojuego agrupa los datos históricos en un array de hasta 255 equipos, donde se espera que cada elemento del array permita acceder al conjunto de registros históricos de cada equipo ordenados por el campo **fecha**. El archivo posee registros con los datos de todos los partidos jugados por cada equipo. Cada registro de partido posee la siguiente estructura:

id_equipo	**id_partido**	**fecha**	**puntos**	**jugadores**
entero (0-254)	entero	entero largo	entero	array de enteros [18]

El campo **fecha** contiene el resultado de llamar a la función time(NULL) de la biblioteca time.h, por lo tanto es un valor entero desde donde es posible obtener la secuencia de partidos.

Se pide: Crear la función ***cargarHistorico()*** que recibe por parámetro, el array (con todos los elementos inicializados en NULL), la ruta al archivo y deberá: 1) leer y agrupar todos los registros de partidos por **id_equipo**, 2) enlazar todos los partidos ascendentemente (0-9) según el valor del campo **fecha**, siendo que los registros pueden haber sido guardados en el archivo sin contemplar ningún tipo de orden. Crear las estructuras necesarias.

(50 puntos)

Solución

1. **Algoritmo**: [X] Intercala los nodos de dos listas. Comentario: Es una implementación del algoritmo apareo / merge.

2. **Chat.**

```
struct Cliente {
  int id_cliente;
  string nickname;
};

void agregarCliente(Nodo<Cliente>*& p, Nodo<Cliente>*& q, Cliente c){
  encolar(p,q,c);
}

Cliente atenderProximo(Nodo<Cliente>*& p, Nodo<Cliente>*& q){
  return desencolar(p,q);
}

int main() {
  Nodo<Cliente>* p = NULL;
  Nodo<Cliente>* q = NULL;

  Cliente cli1;
  cli1.id_cliente = 1;
  cli1.nickname = "Zae";
  agregarCliente(p,q,cli1);

  Cliente cliente = atenderProximo(p,q);
  cout << "Atendiendo a: " << cliente.nickname << " con ID: " << cliente.id_cliente << endl;

  return 0;
}
```

3. **Videojuego.**

```
const unsigned CANT_EQUIPOS = 255;

struct Partido {
  int id_equipo;
  int id_partido;
  long fecha;
  int puntos;
  int jugadores[18];
};

int criterioFecha09(Partido p1, Partido p2) {
  return p1.fecha - p2.fecha;
}

void cargarHistorico(Nodo<Partido>* equipos[CANT_EQUIPOS], char ruta[]) {
  FILE* file = fopen(ruta,"rb+");
  Partido p;
  fread(&p,sizeof(Partido),1,file);
  while( !feof(file) ) {
    insertarOrdenado<Partido>(equipos[p.id_equipo],p,criterioFecha09);
    fread(&p,sizeof(Partido),1,file);
  }
  fclose(file);
}
```