Guia de trabajos prácticos N° 6

Assembler máquina virtual

1. Indicar en los siguientes casos de invocación a funciones, como sería la invocación a la subrutina en ASM de la MV. Considerar que las variables a, b y c se encuentran en las posiciones de memoria BP-4, BP-8 y BP-12, respectivamente, y corresponden a variables locales de la función que invoca.

   a. `proce(a, b, c);`

   b. `c = func(*a, b);`

   c. `proce (&a, *b, *c);`

2. Dados los siguientes algoritmos. Transcribirlos en ASM de la MV e indicar cómo quedaría la pila al final de su ejecución.

a.

```
int j=3;

void main(){
  int arreglo[5];
  arreglo[0] = 1020;
  arreglo[1] = arreglo[0] | 0x3FF;
  calculo( j, arreglo);
  }
void calculo( int x, int vec[]){
  vec[x] = vec[x-1] & vec[0];
  }
```

Nota: el arreglo ***arreglo[]*** se ubica entre las variables locales del main, y ***j*** es la única variable global definida.

b.

```
int vec[9]

void main(){
  int i=2;
  vec[i] = 501;
  vec[0] = 0xF0F0 & vec[i];
  proce( i, vec );
  }
void proce( int x, int w[]){
  w[x-1] = w[x] | w[0];
  }
```

Nota: el arreglo ***vec[]*** es la única variable global.

3. Transcribir a assembler el siguiente código:

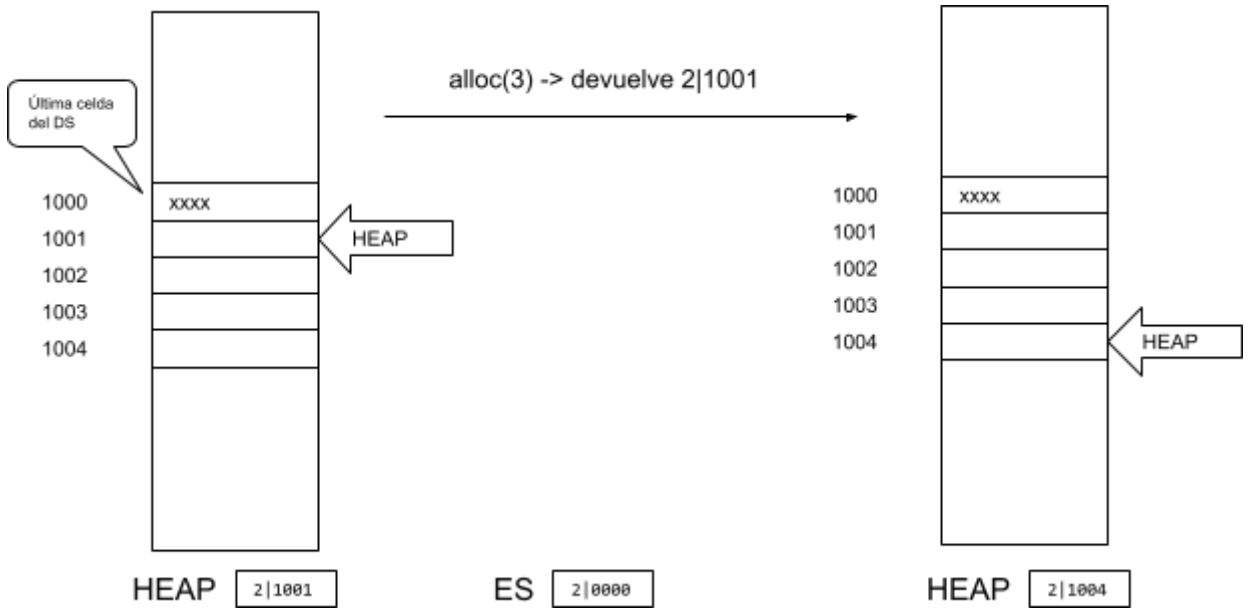
```
void main(){
    int* z = malloc(sizeof(int));
    int vec[] = malloc(sizeof(int) * 100);
    *z = 100;
    funcion1(vec, *z);
}

int  functionl(int vec[], int z) {
    int s;
    int i;
    vec[0] = 1;
    for (i = 1; i < z; i++)
        vec[i] = vec[i-1] * i;
    s = sum(vec, &z);
    print_int(s);
}
```

```
int sum(int vec[], int* n)
{
    static int s = 0;
    register int i;
    for (i = 0; i < *n; i++)
        s = s + vec[i];
    return s;
}
```

NOTA: malloc devuelve las posiciones de memoria con offset 4 y 8 del *Extra Segment*.

4. Crear una estructura “heap” para reservar memoria dinámicamente.
   La estructura funciona de la siguiente manera:

   a. Se utilizan los primeros 4 bytes del *Extra Segment* (**HEAP**) para almacenar el puntero (doble indirección) a la primera celda disponible del *Extra Segment* para memoria dinámica, que inicialmente será la siguiente.
   b. Crear una subrutina **alloc(n)** donde n -pasado por parámetro- es la cantidad de bytes de memoria que se quiere reservar.
   c. **alloc(n)** devuelve la dirección de memoria solicitada en el registro EAX e incrementa el contenido de la celda HEAP según “n” para apuntar al siguiente espacio de memoria disponible.



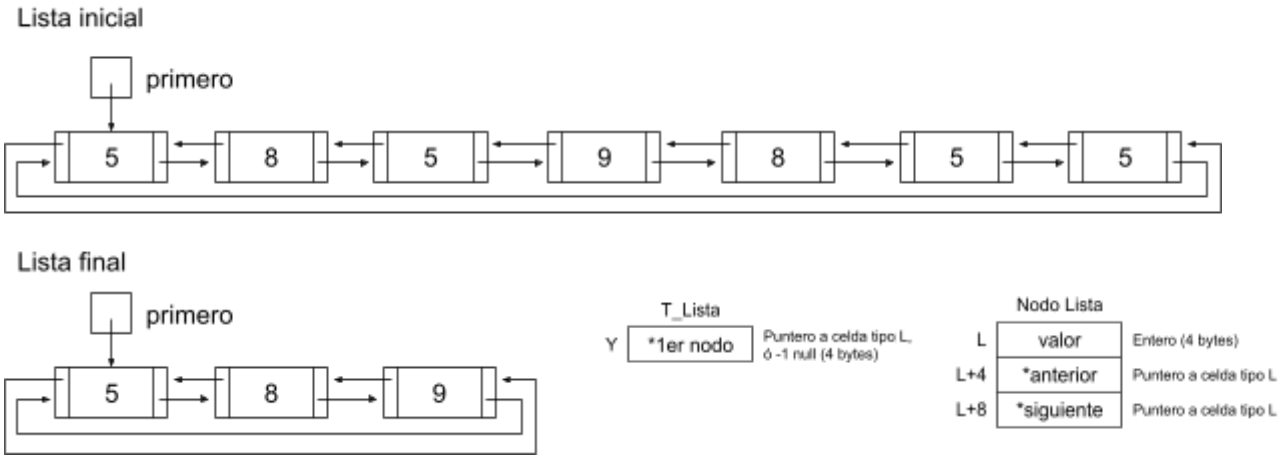
5. Utilizando alloc, crear subrutinas que permitan administrar listas dinámicas simplemente enlazadas de números enteros:
   a. Crear nodo
   b. Insertar nodo en forma ordenada
   c. Remover nodo con valor pasado como parámetro
   d. Buscar nodo
   e. Imprimir la lista

   La lista tiene la siguiente estructura:

6. Escribir una subrutina que reciba dos listas de números (positivos, ordenados en forma creciente) y las intercale formando una lista ordenada.
   a. Creando una nueva lista.
   b. Reutilizando los mismos nodos (modificando las listas originales).

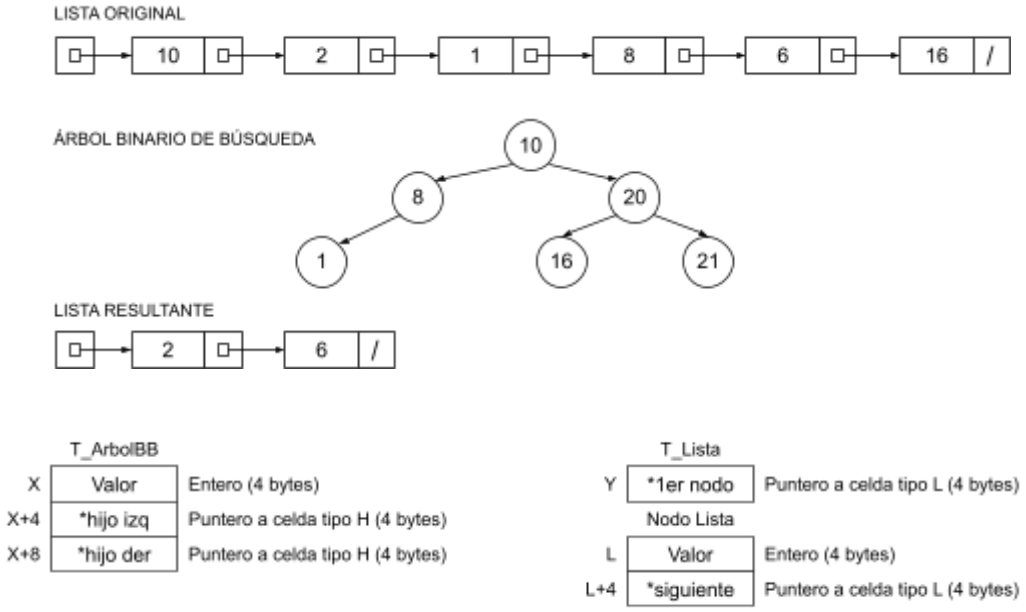
7. Crear una subrutina que invierta el orden de una lista simplemente enlazada.

8. Eliminar de una lista dinámica circular doblemente enlazada, todos los nodos con valor repetido. Además debe devolver la cantidad de nodos eliminados.



9. Crear una estructura de árbol binario que permita cargar ordenados números ingresados por el usuarios (indicando el último número con un número negativo). Armar subrutinas para:
   a. Agregar un nodo al árbol.
   b. Mostrar el árbol inorden.
   c. Mostrar el árbol preorden.
   d. Mostrar el árbol postorden.
   e. Solicitar un número e informar si el mismo se encuentra o no en el árbol.
   f. Crear una subrutina que indique la cantidad de niveles que tiene el árbol.

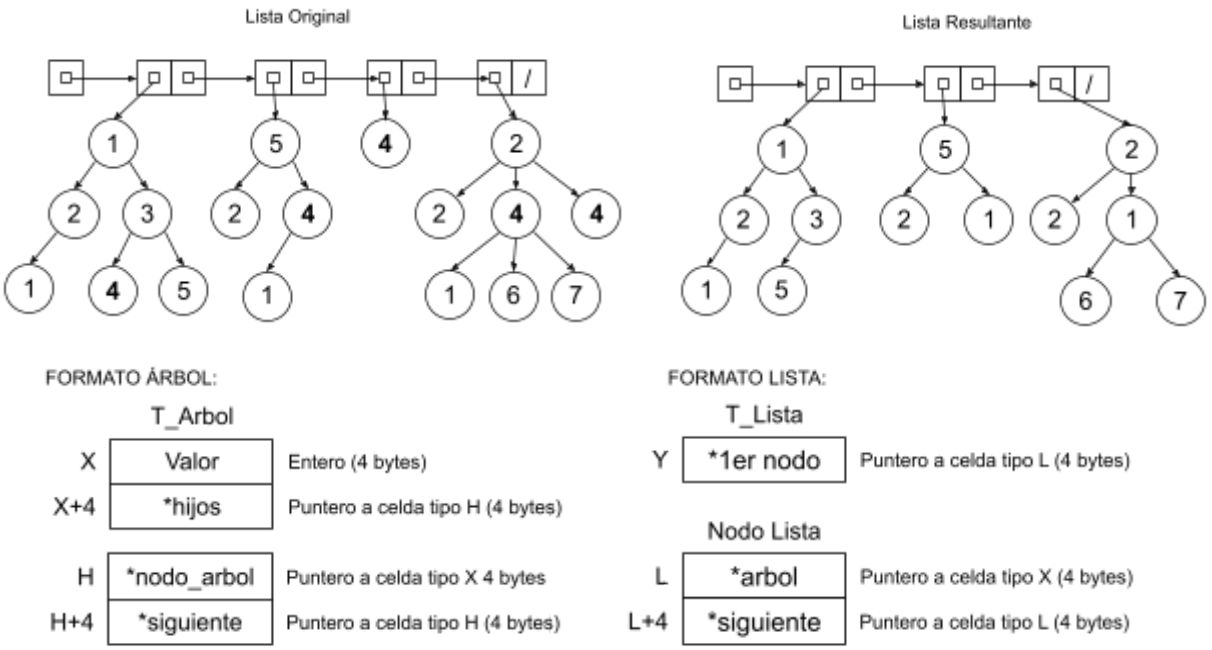
10. Dada una lista dinámica simplemente enlazada y un árbol binario de búsqueda, eliminar de la lista todos los nodos cuyo valor se encuentre en el árbol.



11. Dado un valor entero: eliminar de una lista dinámica simplemente enlazada, con punteros a árboles n-arios, todos los nodos de cada árbol con dicho valor. Cada nodo es reemplazado por su primer hijo. Además debe devolver la cantidad de nodos eliminados.

    Debe tener en cuenta que si un nodo de la lista queda apuntando a un árbol vacío el mismo debe ser eliminado.

    Ejemplo: valor de entrada 4, valor de salida 5, para la siguiente lista original:



12. Contar la cantidad de caminos entre un vértice origen y uno destino de un digrafo acíclico.

    Ejemplo: