

**EXAMEN**

**Segundo Departamental**

**Instituto Politécnico Nacional**

**Escuela Superior de Cómputo**

**(ESCOM)**

**Grupo: 2CM1**

**Profesor: Tecla Parra Roberto**

**Materia: Algoritmos y Estructura de Datos**

**Alumno: Díaz Amaro Josué Ramsés**

**Fecha: 29/05/2025**

**2a GUIA ESTRUCTURA DE DATOS**

Nombre: Grupo:

1.-En un(a) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_el 1er elemento en entrar es el primero en salir

|  |  |
| --- | --- |
| a) cola b) arbol c) cola de prioridad d) tabla hash | ( ) |

2.-En una *c*ola de prioridad siempre se elimina el elemento de

|  |  |  |
| --- | --- | --- |
| a) el principio | b) el final | ( ) |
| c) el de mayor (menor) prioridad | d) ninguna de las anteriores |  |

Suponga que los nodos de una lista **doblemente ligada** de enteros se definen en lenguaje C del siguiente modo:

|  |  |
| --- | --- |
|  | struct nodoLD { int info; struct nodoL \*prev, \*sig; };  typedef struct nodoLD NodoLD; |

3.-En los dos códigos de abajo diga a que inciso corresponden

|  |
| --- |
| a) Código para **insertar** unnodo **al inicio** de la lista doblemente ligada |
| b) Código para **insertar el primer** nodode la lista doblemente ligada |
| c) Código para **insertar después** de un nodo de referencia doblemente ligada |
| d) Código para **insertar antes** de un nodo de referencia doblemente ligada |

|  |  |
| --- | --- |
| p=(NodoLD\*) malloc(sizeof(NodoLD) );  p->info=INFO;  p->sig=NULL;  p->prev=NULL;  head = p;  tail = p; | p=(NodoLD\*) malloc(sizeof(NodoLD) );  p->info=INFO;  p->sig= head;  p->prev=NULL;  head->prev = p;  head = p; |
| Se recorre la lista hasta encontrar el nodo de referencia, que queda apuntado por q y p apunta al nuevo nodo,  p=(NodoLD\*) malloc(sizeof(NodoLD) );  p->info=INFO;  p->sig = q->sig;  p->prev = q;  q->sig->prev = p;  q->sig = p; **Respuesta**  ( ) | Se recorre la lista hasta encontrar el nodo de referencia, que queda apuntado por q p apunta al nuevo nodo.  p=(NodoLD\*) malloc(sizeof(NodoLD) );  p->info=INFO;  p->sig = q;  p->prev = q->prev;  q->prev->sig = p;  q->prev = p; **Respuesta** ( ) |

Suponga que los nodos de una lista **simplemente ligada** de enteros se definen en lenguaje C del siguiente modo:

struct nodoL { void \*info; struct nodoL \*sig; }

typedef struct nodoL NodoL;

4.-En los dos códigos de abajo diga a que inciso corresponden

|  |
| --- |
| a) Código para **insertar** en una una cola circular |
| b) Código para **eliminar** de una cola circular |
| c) Código para **crear** un nodo de una lista |
| d) Código para **eliminar** unnodo **al inicio** de la lista |

|  |  |
| --- | --- |
| void func1(NodoL \*\*cola, void \*elem){  NodoL \*p;  p = (NodoL \*)malloc(sizeof(NodoL));  p->info = elem;  if (\*cola == (NodoL \*)NULL)  p->sig = p;  else {  p->sig = (\*cola)->sig;  (\*cola)->sig = p;  }  (\*cola) = p;  } | void func2(NodoL \*\*cola, void \*\*elem){  NodoL \*aux;  \*elem = (\*cola)->sig->info;  if ((\*cola) == (\*cola)->sig) {  \*cola = (NodoL \*)NULL;  }  else {  aux = (\*cola)->sig;  (\*cola)->sig = aux->sig;  }  } |
| NodoL \*func3(NodoL \*\*cab){  NodoL \*p;  if(!\*cab){  return (NodoL \*)NULL;  }  p=\*cab;  \*cab=p->sig;  p->sig=(NodoL \*)NULL;  return p;  } | NodoL \*func4(void \*elem, NodoL \*sig ){  NodoL \*p;  p = (NodoL \*)malloc(sizeof(NodoL));  if(!p){  return (NodoL \*)NULL;  }  p->info = elem;  p->sig =s ig;  return p;  } |

Suponga que los nodos de un **arbol binario** de enteros se definen en **lenguaje C** del siguiente modo:

|  |  |
| --- | --- |
|  | struct nodoArb { int info; struct nodoArb \* izq, \*der; };  typedef struct nodoArb NodoArb; |

Lea el código de las funciones de abajo y luego responda las preguntas (sugerencia haga una prueba de escritorio o un árbol de llamadas en el caso de funciones recursivas). Suponga que los nodos del árbol binario almacenan enteros

|  |  |
| --- | --- |
| NodoArb \*func1(NodoArb \*r, int dato){  while(r && dato != r->info){  if( dato < r->info ) r=r->izq;  else r=r->der;  }  return r; } | void func2(NodoArb \*r){  if( !r ) return;  func2(r->izq);  printf(“%d ”, r->info);  func2(r->der);  } |
| int f3(NodoArb \*r){  if (r == NULL) return 0;  return 1+f3(r->izq)+f3(r->der);  } | int f4(NodoArb \*r){  int a1, a2;  if(r == NULL) return 0;  a1=f4(r->izq); a2=f4(r->der);  if( a1 > a2) return a1+1;  return a2+1;  } |

**Nota:** se puede poner ninguna como respuesta en algunos casos

5.-Cual función devuelve la **cuenta el numero de nodos del árbol** binario?\_\_\_\_\_\_\_\_\_\_\_

6.-Cual función devuelve la **altura del árbol** binario?\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

7.-Cual función **busca un nodo** en un árbol de búsqueda?\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

8a..-Cual función se puede usar para **imprimir inorden** un árbol binario?\_\_\_\_\_\_\_\_\_\_\_\_\_

8b.-Cual función se puede usar para **imprimir preorden** un árbol binario?\_\_\_\_\_\_\_\_\_\_\_\_

8c.-Cual función se puede usar para **imprimir postorden** un árbol binario?\_\_\_\_\_\_\_\_\_\_\_

**prob 1**-Una cola doble o deque (double-ended queue) es un tipo de datos lineal que permite consultar, añadir y eliminar elementos en cualquiera de los dos extremos de la cola. Se pide:

• Implementar las colas dobles usando listas doblemente enlazadas.

**prob 2**-Una cola medieval se comporta como una cola ordinaria, con la unica diferencia de que los elementos almacenados en ella se dividen en dos estamentos: nobles y plebeyos. Dentro de cada estamento, los elementos deben ser atendidos en orden de llegada; pero siempre que haya nobles en la cola, ́estos deben ser atendidos antes que los plebeyos. Se pide:

• Especificar un tipo abstracto de datos para las colas medievales que disponga al menos de operaciones para:

-crear una cola medieval vacía

-añadir un elemento nuevo a una cola,

-consultar el primer elemento de una cola,

-quitar el primer elemento de una cola,

-consultar el numero de nobles en una cola,

-consultar el numero de plebeyos en una cola,

-consultar si una cola es vacía ́ o no.

• Implementar las colas medievales utilizando memoria dinámica.

|  |  |
| --- | --- |
| int f(){  int j, v, max =1 ;  for(j=2; j<= N; j++)  if(a[j] > a[max])  max=j;  v=a[max];  a[max]=a[N--];  return v;  } | int f(NodoArb \*r1, NodoArb \*r2,  int (\*cmp)(void \*, void \*)){  if (r1==NULL || r2==NULL)  return (r1 == r2);  if (cmp(r1->dato, r2->dato)) return 0;  return( f(r1->izq, r2->der, cmp)  && f(r1->izq, r2->der, cmp) );  } |
| NodoArb \*f(NodoArb \*r, void \*dato,  int (\*cmp)(void \*, void \*)){  while(r && cmp(dato,r->dato)){  if(cmp(dato,r->dato)<0)  r=r->izq;  else  r=r->der;  }  return r;  } | NodoArb \*f(NodoArb \*r, void \*dato,  int (\*cmp)(void \*, void \*)){  if (r==NULL) return NULL;  if (!cmp(dato,r->dato)) return r;  if (cmp(dato,r->dato)<0)  return f(r->izq, dato, cmp);  return f(r->der, dato, cmp);  } |
| int f(){  return a == b;  } | void f(int v){  a[N++]=v;  } |
| void f(int dato){  x[b] = dato;  b = (b+1) % N ;  } | int f(){  int t=x[a];  x[a]='\*';  a = (a +1) % N ;  return t;  } |
| int f(NodoArb \*r) {  if (!r) return 0;  if (!r->izq && !r->der)  return 1;  return f(r->izq)+f(r->der);  } |  |

1-Cual función inserta en una cola de prioridad?

2-Cual función elimina en una cola de prioridad?

3.-Cual función **busca recursivamente un nodo** en un árbol de búsqueda?

4.-Cual función que determina si una cola circular esta vacia

5.-Cual función inserta en una cola circular?

6.-Cual función elimina en una cola circular?

7.-Cual función determina si 2 árboles binarios son iguales?

8.-Cual función devuelve la **cuenta el numero de hojas del árbol** binario?

9.-Cual función **busca iterativamente un nodo** en un árbol de búsqueda?

|  |  |
| --- | --- |
| void f(int k) {  int v;  v = a[k]; a[0]= INT\_MAX;  while ( a[k/2] <=v ) {  a[k] = a[k/2]; k = k/2;  }  a[k]= v;  } | void f(int k) { int j, v;  v = a[k];  while ( k <= N/2) {  j = k+k;  if (j < N && a[j] < a[j+1]) j++;  if (v >= a[j]) break;  a[k] = a[j]; k = j;  }  a[k]= v;  } |
| void f(int v) {  a[++N] = v;  upheap(N);  } | int f() { int v = a[1];  a[1] = a[N--];  downheap(1);  return v;  } |
| void f(char s[]) { int k;  for(k = 0; s[k]; k++)  insert(s[k]);  for(k = N; k >= 1; k--)  a[k] = remueve();  } |  |

1.-Cual función inserta en un heap

2.-Cual función elimina en un heap

3.-Cual función ajusta un heap despues de una insercion

4.-Cual función ajusta un heap despues de una eliminacion

5. -Cual función usa un heap para ordenar

|  |  |
| --- | --- |
|  |  |
|  |  |