Auxiliar 4

Estructuras de Datos

Profesores: Alexandra Ibarra, Luis Mateu y Rodrigo Urrea
Auxiliares: Gerard Cathalifaud
Simón Campos
Semestre: Otoño 2024

Resumen

Operador *

El operador * es para acceder al contenido de un puntero (el valor que indica la dirección de memoria). Por ejemplo:

```
int x = 5;
int *p = &x;
int y = *p; // y = 5
```

También se puede cambiar el contenido de lo que apunta, por lo tanto, cambiar el valor de la variable. Continuando con el ejemplo anterior:

```
*p = 10; // Ahora x = 10
```

La utilidad de esto es que, si nosotros queremos que una función modifique una variable, podemos pasar un puntero a esa variable y la función puede modificarla. Por ejemplo:

```
void duplicar(int *p) {
  *p = *p * 2;
}

int x = 5;
duplicar(&x); // Ahora x = 10
```

Malloc y Free

Para utilizar funciones de alocación de memoria, como `malloc` y `free`, es necesario incluir la librería `stdlib.h`. La función `malloc` permite alocar un bloque de memoria de un tamaño específico y retorna un puntero a la dirección de memoria del bloque alocado. Mientras que `free` libera la memoria alocada previamente por `malloc`. Por ejemplo:

```
#include <stdlib.h>

int *p = malloc(sizeof(int)*10); // Alocamos un bloque de memoria para 10 enteros
p[0] = 5;
free(p); // Liberamos la memoria alocada
```

Estructuras

Las estructuras en C permiten agrupar distintos tipos de datos en una sola variable. Por ejemplo:

```
typedef struct p {
  int x;
  int y;
} Punto;

Punto p;

p.x = 5;
p.y = 10;
```

Al crear la estructura con `typedef`, podemos usar `Punto` como un tipo de dato, y no tener que escribir `struct p` cada vez que queramos declarar una variable de tipo `Punto`. Además, podemos acceder a los campos de la estructura con el operador `.`, como en `p.x`.

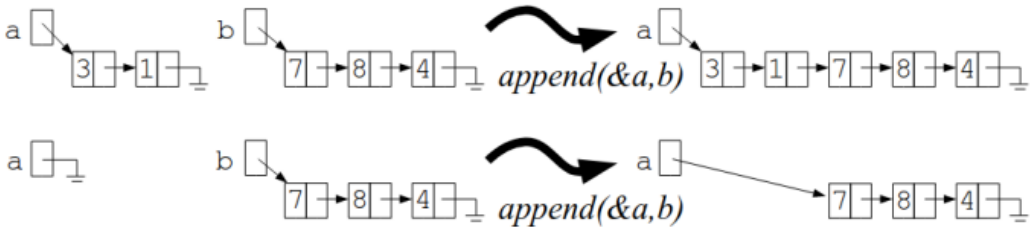
Preguntas

P1. Programe la función append que adjunta la lista enlazada en b a la lista *pa. El encabezado de la función es el siguiente:

```
typedef struct nodo {
  int x;
  struct nodo *sgte;
} Nodo;

void append(Nodo **pa, Nodo *b);
```

La función debe ser eficiente, es decir, su tiempo de ejecución debe ser proporcional al tamaño de la lista *pa. No puede usar ciclos (como `while` o `for`). Debe usar recursión. No puede usar `malloc`. Reutilicen los punteros recibidos y mantengan el orden. Ejemplo en la siguiente figura:



P2. [**Propuesto**] Implementar las cola **FIFO** y pila **LIFO** utilizando listas enlazadas. Para esto, se debe implementar las funciones `push`, `pop` y `peek` para la pila, y `enqueue`, `dequeue` y `peek` para la cola. Las estructuras de las listas enlazadas son las siguientes:

```c
typedef struct nodo {
  int x;
  struct nodo *sgte;
} Nodo;
```

Las funciones deben tener la siguiente firma:

```c
Cola *crearCola();
void enqueue(Cola *cola, int x);
int dequeue(Cola *cola);
int* peek(Cola *cola);
void liberarCola(Cola *cola);

Pila *crearPila();
void push(Pila *pila, int x);
int pop(Pila *pila);
int* peek(Pila *pila);
void liberarPila(Pila *pila);
```

`peek` debe retornar el valor del primer elemento de la estructura sin eliminarlo.

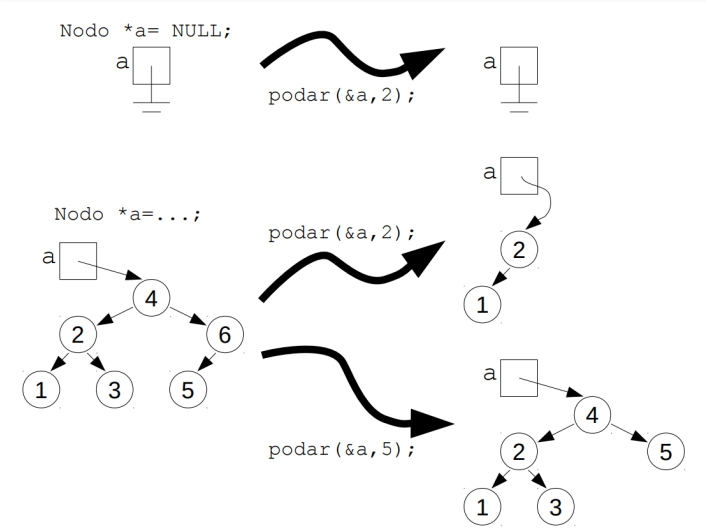
P3. **(P2 C1 Primavera 2017)** Sea la estructura:

```c
typedef struct nodo {
  int x;
  struct nodo *izq, *der;
} Nodo;
```

Programe la función:

```c
void podar(Nodo **pa, int y);
```

Esta función debe modificar un árbol de búsqueda binaria `*pa` eliminando todos los nodos etiquetados con valores mayores que `y`. No necesita liberar la memoria de los nodos eliminados. Estudie los 3 ejemplos de uso de la siguiente figura:



Restricciones: Sea eficiente, el tiempo de ejecución debe ser proporcional a la altura del árbol en el peor caso. No puede usar ciclos (como `while` o `for`). Debe usar recursión. No puede usar `malloc`.

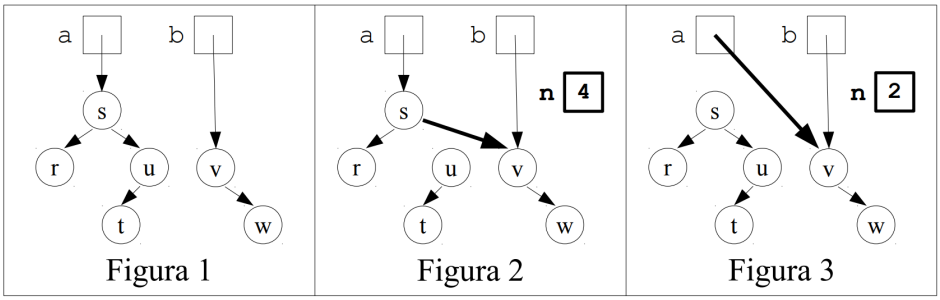
P4. **[Propuesto] (P2 C1 Otoño 2017)** Sea la estructura:

```
typedef struct nodo {
  char c;
  struct nodo *izq, *der;
} Nodo;
```

Programe la función:

```
int reemplazarNodoK(Nodo **pa, int k, Nodo *b);
```

Sea `a = *pa`. Esta función reemplaza el k-ésimo nodo del árbol a por el nodo b. El k-ésimo nodo de a es el k-ésimo nodo al enumerar los nodos de a en inorden. Por ejemplo en la figura 1, r es el nodo 1, s el 2, t el 3 y u el 4. Esta función retorna k si se hizo el reemplazo, es decir cuando el árbol a tenía al menos k nodos. Si no, entrega el número de nodos encontrados en a, que será inferior a k. Las siguientes figuras sirven para explicar algunos ejemplos de uso. Los punteros a y bE son de tipo `Nodo*`.



La figura 2 se obtiene cuando a partir de la figura 1 se llama:

```
int reemplazarNodoK(&a, 4, b);
```

Se reemplazó u por v y la función retornó 4. La figura 3 se obtiene cuando a partir de la figura 1 se llama:

```
int reemplazarNodoK(&a, 2, b);
```

Acá se reemplazó la raíz s del árbol por v. Por eso se requiere que el puntero a la raíz del árbol se pase por referencia (`&a`). Por último si a partir de la figura 1 se intentara reemplazar el quinto nodo de a que no existe, no se haría ningún reemplazo y la función retornaría 4.