Programación de software de sistemas

Ayudantía 4

Profesor: Rodrigo Verschae

Ayudante: Nicolás Araya

Sintaxis fopen() de stdlib.h

```
FILE *fopen(const char *filename, const char *mode)
```

1	**"r"** Opens a file for reading. The file must exist.
2	**"w"** Creates an empty file for writing. If a file with the same name already exists, its content is erased and the file is considered as a new empty file.
3	**"a"** Appends to a file. Writing operations, append data at the end of the file. The file is created if it does not exist.
4	**"r+"** Opens a file to update both reading and writing. The file must exist.
5	**"w+"** Creates an empty file for both reading and writing.
6	**"a+"** Opens a file for reading and appending.

Ejemplo escritura y lectura

```c
#include <stdio.h>
#include <stdlib.h>

int main () {
   FILE * fp;

   fp = fopen("file.txt", "w+");
   fprintf(fp, "%s", "HOLA");

   fclose(fp);

   return(0);
}
```

```c
#include <stdio.h>
#include <stdlib.h>

int main () {
   FILE * fp;
   int c;
   fp = fopen("file.txt", "r");
   while((c=fgetc(fp)) != EOF)
   {
       printf("%c", c);
   }
   printf("\n");
   fclose(fp);

   return(0);
}
```

Syntaxis fread() de stdlib.h

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream)
```

void *ptr: Puntero a bloque de memoria con un tamaño mínimo de tamaño*nmemb bytes.
size_t size: El tamaño en bytes de cada elemento a leer.
size_t nmemb:El número total de elementos, cada uno de tamaño “size” bytes.
FILE *stream: Puntero al objeto FILE.

Ejemplo fread()

```c
#include <stdio.h>
#include <string.h>

int main () {
   FILE *fp;
   fp = fopen("file.txt", "r");
   fseek(fp, 0, SEEK_END);
   int file_size = ftell(fp);
   fseek(fp, 0, SEEK_SET);
   char *string = malloc(file_size * sizeof(char));

   fread(string, sizeof(char), file_size, fp);

   printf("%s\n", string);
   fclose(fp);

   return(0);
}
```

Macros

Cree 2 macros EOC y ISNUM que reciban un argumento y retornen:

End Of String: 1 si está en el último char del string, 0 caso contrario.

IS NUMber: El char transformado en entero si el char corresponde a un carácter numérico, 0 caso contrario.

Solución

```
#define EOC(x) (((x) != '\0') ? 1 : 0)
#define ISNUM(c) (((c) >= '0' && (c) <= '9') ? \
                                    (c) - '0' : 999)
```

Macro Funcional

Cree una macro funcional que realice un SWAP a 2 variables de cualquier tipo.

Solución

```
#define SWAP(x, y, T) \
        do { \
                T temp = x; \
                x = y; \
               y = temp; \
        } while (0)
```

Funciones Variádicas

Cree una función double average(int count, …) que retorne el promedio de todas las entradas.

Solución

```c
double average(int count, ...)
{
    va_list ap;
    int j;
    double sum = 0;

    va_start(ap, count);
    for (j = 0; j < count; j++)
    {
        sum += va_arg(ap, int);
    }
    va_end(ap);

    return sum / count;
}
```

Funciones Variádicas

Cree una función llamada `char *concatenarVariadica(const char *separator, …)` que concatene los Strings que reciba con el separador especificado.

Solución parte 1

Encabezado:

```c
#include <stdio.h>
#include <stdarg.h>
#include <string.h>
#include <stdlib.h>
```

```c
char *concatenarVariadica(const char *separator, ...) {
        va_list args;
        va_start(args, separator);

        int size = 0;
        const char *arg;
        while ((arg = va_arg(args, const char *)) != NULL) {
            size += strlen(arg);
           if (separator != NULL) {
               size += strlen(separator);
           }
       }
```

Solución parte 2

```
    char *result = (char *)malloc(size + 1);
    if (result == NULL) {
        return NULL;
    }

    va_start(args, separator);
    char *current = result;
```

Solución parte 3

```
    while ((arg = va_arg(args, const char *)) != NULL) {
        size_t len = strlen(arg);
        memcpy(current, arg, len);
        current += len;
        if (separator != NULL) {
            len = strlen(separator);
            memcpy(current, separator, len);
            current += len;
        }
    }
    *current = '\0';

    va_end(args);

    return result;
}
```

Struct

```
struct persona {
    int edad;
    float altura;
};
```

```
struct persona p;
p.edad = 25;
p.altura = 1.78;
```

```
struct persona *ptrPersona = &persona1;
ptrPersona->edad = 30;
```

typedef struct

typedef es una palabra reservada (keyword) que permite asignarle un alias a un tipo de datos.

```
struct persona {
    int edad;
    float altura;
};
typedef struct persona Persona;
```

```
typedef struct persona {
    int edad;
    float altura;
} Persona;
```

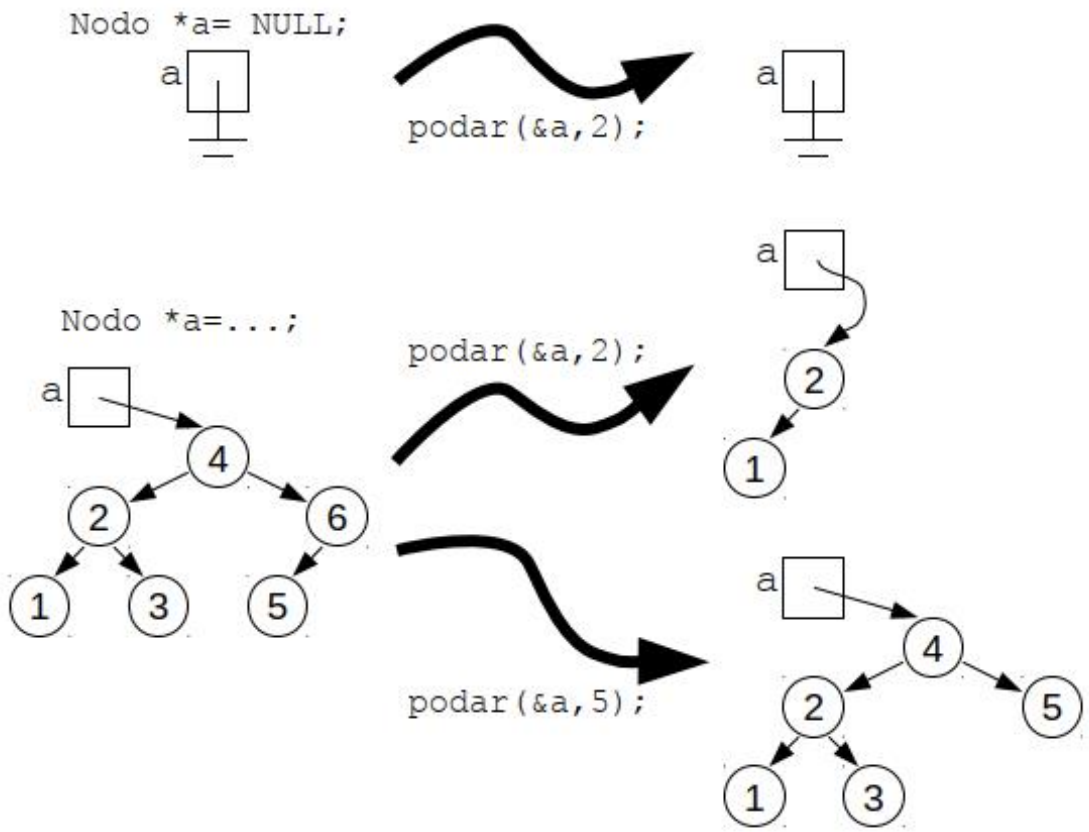
Pregunta 2 (40%)

Programe la función:

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

Esta función debe modificar el 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 figura de la derecha.

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*.



Solución en github.

Es muy grande para poner aquí ^^

Desafío Macros

La empresa computines y asociados ha pedido crear un archivo tipo header: debug.h. Este archivo, mientras se encuentre en la carpeta del código main.c podrá ser importado con #include “debug.h”. Debe poder imprimir mensajes y funcionar como un printf, ósea printf(“STRING” “STRING” “%d\n”, x).

Modo de uso:

```c
int main() {
    debug_message(DEBUG, "This is a debug message.");
    debug_message(INFO, "This is an info message.");
    debug_message(ERROR, "This is an error message.");

    return 0;
}
```

```
[DEBUG] In file: test.c, line: 6: This is a debug message.
[INFO] In file: test.c, line: 7: This is an info message.
[ERROR] In file: test.c, line: 8: This is an error message.
```

Para crear el archivo usted dispone de las siguientes herramientas:

“Macros Predefinidas”: __FILE__, __LINE__, __VA_ARGS__.

“__VA_ARGS__”: Corresponde a una lista de los argumentos.

“Sintaxis especial de macros funcionales” (Pequeño spoiler):

level: DEBUG, INFO o ERROR

format: El string/mensaje que el usuario quiere ver

```
#define debug_message(level, format, ...) \
        //linea 1 \
        //linea n \
        //ultima linea
```

“printf”: Puede llamar a printf en la macro funcional. Plus: printf concatena los strings para imprimirlos en pantalla.

“DO-WHILE”: Debe tener cuidado al llamar funciones en las macros.

“Inclusión multiple”: Debe controlar cuando el .h sea llamado multiples veces. Pista: #ifndef MACRO

EXITOOOOOO