# ÍNDICE

# VII
# CLASE 7

Estos son los apuntes no oficiales del curso CC3301-1 cursado en Otoño 2022 bajo las cátedras del docente Luis Mateu B. El material usado es directamente obtenido de cátedras, en adición a los materiales del curso. [1].

## 1 Esquema de la clase

- Tiempo de vida de una variable
- Variables locales
- La pila de registros de activación
- Variables dinámicas malloc/free
- El heap de memoria
- Errores comunes: Memory leak y dangling
- Sanitize y valgrind

## 2 Problema: Función que retorna una copia de un string

```c
char *copia(char*str) {
    char res[strlen(str)+1];
    return strcpy(res, str); // NO HAGA ESTO
}
```

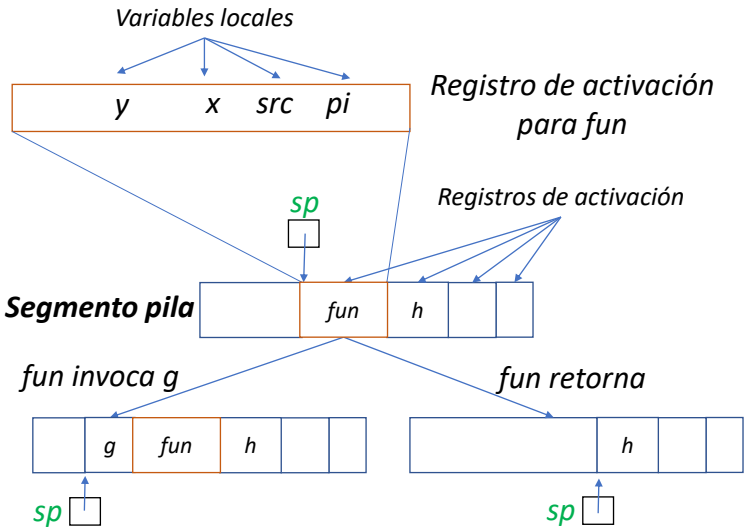
Código 1: ¡`strcpy` retorna `res`!

- Las variables declaradas dentro de una función se llaman variables locales o automáticas, por ejemplo `res` y `str`
- Los arreglos y estructuras también son variables
- Definición: tiempo de vida de una variable

  desde que se crea la variable

  hasta que se destruye
- La creación significa que la variable ya tiene atribuida una dirección en memoria y por lo tanto se puede usar
- La destrucción de una variable significa que la memoria que ocupaba se puede atribuir a otras variables y por lo tanto ya no se debe usar
- Las variables locales se crean en el momento de su declaración y se destruyen automáticamente cuando se sale del bloque en donde fueron declaradas

## 3 Implementación




- sp es el puntero al tope de la pila (stack pointer)
- Para apilar un registro de activación basta sumar su tamaño a sp
- Para desapilarlo hay que restar su tamaño a sp
- Crear/destruir variables locales tiene costo 0 en tiempo de ejecución

Sin la pila no sería posible la recursividad

- La creación y destrucción es solo conceptual
- Por razones de eficiencia se atribuye la memoria a las variables locales antes de su declaración
- El espacio de las variables locales se atribuye casi siempre al inicio de la función en un área de memoria denominada registro de activación o frame, casi siempre de tamaño fijo
- Las variables locales ocupan un lugar fijo en el registro de activación
- Al inicio de una función se apila su registro de activación en el segmento pila
- Se desapila en el momento del retorno
- La función copia es incorrecta porque se usa el arreglo local res después de su destrucción
- ¿Por qué no siempre? La excepción son las funciones que declaran arreglos de tamaño variable

## 4 Variables dinámicas

- Son variables que se crean al invocar la función malloc y se destruyen explícitamente con la función free
- Ejemplo: malloc(20)
- Crea una variable dinámica de 20 bytes
- La variable dinámica no tiene identificador
- No tiene tipo todavía
- La función malloc retorna su dirección que debe almacenarse en un puntero
- El tipo del puntero determina el tipo de la variable dinámica
- Ejemplo: char *str = malloc(20);
- La variable de 20 bytes será un arreglo de 20 caracteres que podría almacenar un string
- str es el identificador del puntero, no de la variable dinámica
- Si se destruye str, no se destruye la variable dinámica
- La función malloc es imprescindible para implementar las estructuras de datos recursivas como los ABBs

## 5 Implementación correcta de la función que retorna una copia de un string

```
// Versión legible
char *copia(char *str) {
    char *res= malloc(strlen(str)+1);
    strcpy(res, str);
    return res;
}
```

```
// Versión ilegible, pero correcta
char *copia(char *str) {
return strcpy( malloc(strlen(str)+1), str );
}
```

## 6 Destrucción de variables dinámicas

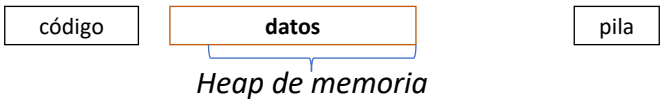
- La destrucción de las variables dinámicas es explícita invocando la función `free`
- Recibe como parámetro la dirección de la variable dinámica
- Ejemplo: `char *str=malloc(10); ... free(str);`
- Una variable dinámica se crea con malloc y vive hasta que se destruye con free
- La dirección que recibe free debe haber sido retornada previamente por malloc variable dinámica

```
int a; \st{free(&a)};
int *p=malloc(5*sizeof(int)); ... ; \st{free(&p[4])};
free(p); // correcto
```

La destrucción es solo conceptual: significa que a partir de ese instante la función malloc puede atribuir esa misma memoria para una nueva

## 7 El heap de memoria

- Las variables dinámicas se alojan en el segmento de datos del programa, en un área que se denomina el memory heap
- Heap significa montón
- No tiene nada que ver con la estructura de datos heap que estudiaron en el curso de algoritmos




- La funciones malloc y free administran el heap de memoria
- `malloc(n)` **busca** un área de memoria contigua de n bytes que esté disponible: crear una variable dinámica tiene un sobrecosto en tiempo de ejecución
- Antes de entregarla, registra su tamaño y la marca como ocupada
- En `free(dir)` el área de memoria que comienza en dir debe estar marcada como ocupada todavía
- El área quedará marcada como disponible

```
free(dir); ...  free(dir); // incorrecto
```

- La función malloc de C cumple la misma función que el operador new de Java (y C++)
- No hay un recolector de basuras en C: **en C hay que destruir las variables dinámicas explícitamente con free**
- Cuando no se invoca free para destruir una variable dinámica que ya no se necesita, esa variable se transforma en un memory leak o goteo de memoria

- La función malloc nunca reutilizará esa memoria
- Cuando se agota la memoria disponible en el heap, malloc solicita al núcleo del sistema operativo la extensión del segmento de datos, haciendo crecer el heap
- En los programas que poseen memory leaks el heap no para de crecer, hasta que el núcleo ya no puede extender el segmento de datos y malloc retorna NULL: la dirección 0
- Típicamente el programa se cae por segmentation fault
- Cuando un programa termina, el núcleo destruye todos sus segmentos, liberando toda la memoria, incluyendo goteras de memoria

## 8 Referencias colgantes

- Cuando una variable es destruida, es válido que su dirección quede almacenada en un puntero
- Esa dirección se denomina *dangling reference* o referencia colgante
- Por ejemplo la dirección retornada por la función copiar del inicio de esta clase es una referencia colgante
- Acceder al contenido de una referencia colgante es un error porque tal vez esa memoria ya fue atribuida a otra variable y fue modificada
- Al liberar una variable dinámica con `free(ptr)`, la dirección almacenada en ptr no cambia y por lo tanto es una referencia colgante
- Mientras malloc no atribuya esa memoria a otra variable dinámica, el contenido de la variable dinámica no va a cambiar
- Es un error acceder la contenido de ptr o una copia de ptr

## 9 Herramientas para codecaptionubrir errores de manejo de memoria

### 9.1 Sanitize de gcc (y clang)

- la opción `-fsanitize=address` de gcc genera código adicional para detectar goteras de memoria y uso de referencias colgantes
- En contrapartida el programa corre mucho más lento
- No detecta todos los errores, ¡sea cuidadoso!

### 9.2 Valgrind

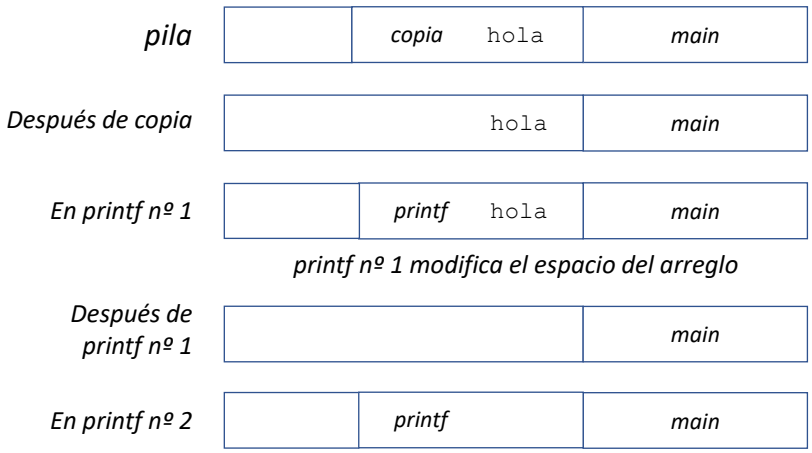
- El comando `valgrind` instrumenta un binario ejecutable compilado con gcc agregando mucho código para verificar el uso correcto de *malloc* y *free* detectando goteras de memoria y uso referencias colgantes asociadas a malloc
- En contrapartida el programa corre aún más lento que con sanitize
- Detecta menos errores que sanitize

## 10 El recolector de basuras

- Un recolector de basuras libera automáticamente las variables dinámicas que ya no son alcanzables por el programa
- C y C++ no posee recolector de basuras
- *Java* y *Python* sí poseen recolector de basuras
- No hay manera de liberar explícitamente la memoria
- La ventaja es que no pueden haber errores asociados a dangling references
- Casi no hay *memory leaks*
- No se pierde tiempo de desarrollo en codecaptionubrir en donde se debe liberar la memoria
- Pero el recolector de basuras introduce un sobrecosto importante en tiempo de ejecución y uso de memoria

- Las implementaciones más eficientes introducen pausas en la ejecución que son molestas en aplicaciones interactivas
- Hay recolectores de basuras que minimizan las pausas pero con un sobrecosto adicional en tiempo de ejecución

## 11 Explicación del error en copiar



# REFERENCIAS

[1] Mateu, Luis: *Programación de Software de Sistemas - Novedades.* `https://www.u-cursos.cl/ingenieria/2022/1/CC3301/1/novedades/`, 2022.