ÍNDICE

VI
CLASE 6

Estos son los apuntes no oficiales del curso CC3301-1 cursado en Primavera 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 Agenda 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 leaks y dangling referencese
- Sanitize y valgrind

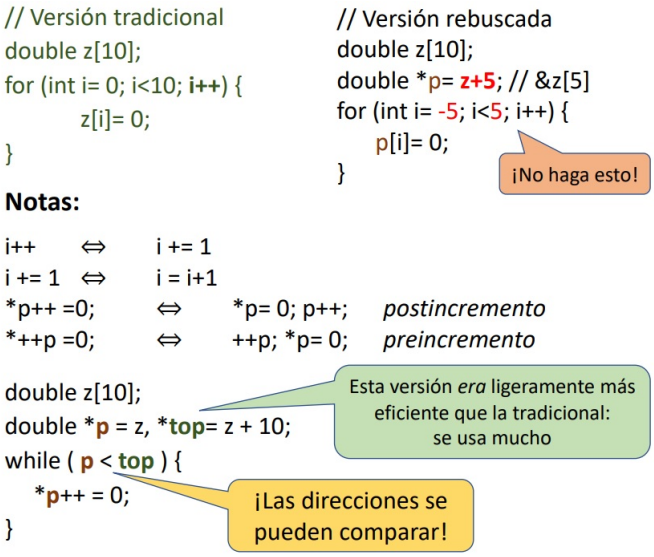
2 Resumen de la clase pasada: Variables y punteros

- Una variable reside en la memoria
  - Almacena valores de un tipo específico
  - Se puede declarar, asignar y evaluar
  - También se puede obtener su dirección
  - Y su tamaño en bytes
- Un puntero es una variable que almacena direcciones de variables
  - Sirven para implementar los strings, las estructuras de datos y mucho más
  - El operador de contenido permite acceder a la variable a la cual apunta un puntero
- Un arreglo es un conjunto enumerado de variables del mismo tipo
  - El operador de subindicación permite seleccionar una de esas variables por medio de un índice
  - Hay una equivalencia entre subindicación y aritmética de punteros

3 Aritmética de punteros

Visitar Clase 5, sección 7

4 Maneras rebuscadas de inicializar un arreglo en 0, pero correctas



5 Cuidado con los arreglos

Visitar Clase 5, sección 7

6 Más sobre aritmética de punteros

- Si p y q son punteros o arreglos, las siguientes expresiones son inválidas: `p*5` `p+2.5` `p/10` `p+q`
- Solo tiene sentido `p + o - una expresión entera`
- Si p y q son punteros del mismo tipo:

  `p - q` es correcto y es de tipo entero
- El valor de p − q satisface:

  `p - q = i` ⇔ `p = q + i`

7 Más sobre arreglos y punteros

Visitar Clase 5, sección 10

8 Strings

- Un string es un arreglo de caracteres que termina con un byte que almacena el valor 0: no `'0'`
- Cuidado: $48 = \texttt{'0'} \neq 0$
- Ejemplo: `char str[ ] = {'H', 'o', 'l', 'a', 0};`
- Se referencian por medio de la dirección de su primer caracter
- Ejemplo: `printf("%s\n", str );`
- Se puede asignar a un puntero: char *r = str;

Ejemplo: contar las letras mayúsculas:

```
char *r = str;
int cnt = 0;
while (*r != 0) {
if ('A' <= *r && *r <= 'Z')
cnt++;
```

```
r++;
}
printf("Mayúsculas: %d\n", cnt);
```

8.1 Strings constantes

- Todo lo que se escribe entre `" ... "`
- Ejemplo: `char *str2 = "Hola";`
- ¡No se pueden modificar! `*str2 = 'h'; // Seg. Fault`
- Se almacenan en un área de memoria de solo lectura

Sintaxis especial para declarar strings mutables:

```
char str3[ ] = "Hola"; // No estaba en el C original
*str3 = 'h'; // Correcto
printf("%d\n", str3); // Muestra hola, h minúscula
str3 [ ] = "Hello"; // Error sintáctico
```

8.2 Funciones para manipular strings

- `int strlen(char *s)`: calcula el largo de un strings, sin contar el 0 que lo termina
- `strlen("Hola")`: es 4, no entrega el tamaño de memoria atribuido
- `char *strcpy(char *d, char *s)`: copia el string s en el string d

```
char d[20];
strcpy(d, "Hola");
```

Código 1: El destino d debe ser la dirección de un área de tamaño suficiente (largo del string + 1)

```
char *p;
strcpy(p, "Hola"); // Incorrecto, ¿seg. Fault?
```

Código 2: (Incorrecto) Porque p no ha sido inicializado con ninguna dirección válida

```
char s[ ] = "Hola"; |$\iff$|  char s[strlen("Hola")+1];
                              strcpy(s, "Hola")
```

8.3 Comparación de strings

`int strcmp(char *s, char *r)`: compara los strings s y r retornando 0 si son iguales, < 0 si s es lexicográficamente menor que r y > 0 si es mayor

```
char s[20]= "juan";
strcmp(s, "pedro") // es < 0
strcmp(s, "diego") // es > 0
strcmp(s, "juan") // es 0
strcmp(s, "Juan") // es > 0
```

- Cuidado con los operadores relacionales `== != < > <= >=` porque comparan direcciones, no contenidos. `s == "juan"` es $\neq 0$

9 Implementación de strlen y strcpy

```c
int mistrlen(char *s) {
    char *r= s;
    while (*r++)
      ;
    return r-s-1;
}
```

```c
char *mistrcpy(char *d, char *s) {
    char *t= d;
    while (*t++ = *s++)
      ;
    return d;
}
```

Nota de Mateu: No promuevo este estilo de código, pero deben aprender a entender este estilo porque hay mucho código en C escrito así.

10 Código usado en clases

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

int mistrlen(char *s) {
  char *r= s;
  while (*r++)
    ;
  return r-s-1;
}

char *mistrcpy(char *d, char *s) {
  char *t= d;
  while (*t++ = *s++)
    ;
  return d;
}

int main() {
  double z[10];
  printf("%p\n", z);
  double *p = z, *top= z+10; // &z[0] y &z[10]
  while ( p < top ) {
    *p++ = 0;
    printf("%p\n", p);
  }

  char str[ ] = {'H', 'o', 'l', 'a', 0};
  printf("%s\n", str);
  char *r = str;
  int cnt = 0;
  while (*r != 0) {
    if ('A' <= *r && *r <= 'Z') {
      cnt++;
    }
    r++;
  }
  printf("Mayusculas: %d\n", cnt);

  char *str2= "Hello";
  // *str2= 'h';
  printf("%s\n", str2);

  char str3[] = "Salut";
  *str3= 's';
  printf("%s\n", str3);

  printf("%ld\n", sizeof(str3));
  printf("%ld\n", strlen(str3));
  r = str3;
  printf("%ld\n", sizeof(r));

  // Ejercicio: ejecutar paso a paso
  // las funciones strlen y strcpy
  printf("%d\n", mistrlen(str3));
  char str4[10];
  mistrcpy(str4, str3);

  return 0;
}
```

REFERENCIAS

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