INGENIERÍA | DEPARTAMENTO
UNIVERSIDAD DE LOS ANDES | DE COMPUTACIÓN
MÉRIDA VENEZUELA | ESCUELA DE SISTEMAS

Laboratorio de Sistemas Operativos
Semestre A-2018
Práctica de Nivelación
Laboratorio

Prof. Rodolfo Sumoza
Prep. Alvaro Araujo
Prep. Luis Sanchez

1 Nivelación de conocimientos

1.1 Punteros en C.

Se puede definir un puntero como una variable que almacena una dirección de memoria, esta dirección de memoria se refiere (o "apunta") a otro valor guardado en otra parte de la memoria. Supongamos que se tiene la variable 'a' la cual es de tipo entero y tiene como valor el número 5, supongamos que se quiere almacenar la dirección de memoria de la variable 'a', para cumplir este objetivo en principio se debería declarar un puntero del mismo tipo de la variable a la que se quiere apuntar, en este caso, de tipo entero, la declaración entonces sería:

```
1    int a = 5;
2    int *pto = NULL;
```

1.1.1 Operadores para punteros.

¿De que manera se asigna la dirección de memoria de la variable 'a' al apuntador? Para responder esta pregunta, se debe introducir el funcionamiento de dos operadores esenciales en el manejo de punteros, ellos serían:

- El Operador '&' (Dirección de) nos permite conocer la dirección de memoria de una variable.

- El Operador '*' (Indirección) nos permite acceder al Valor que contiene la dirección almacenada en el puntero.

Por lo tanto, para asignar la dirección de la variable 'a' al apuntador, se logra:

```
pto = &a;
```

1.1.2 Paso de parámetros a una función.

¿Que formas existen para pasar parámetros a una función?

- Por Valor: es cuando se envía solo el valor que contiene la variable que se quiere pasar por parámetro, por lo tanto, se crea una copia de ese valor en el ámbito de la función que la recibe, en caso de que se modifique la variable pasada por valor en la función destino, no afecta a la variable en la función de donde se invocó esta la función destino.

- Por Referencia: es cuando se envía por referencia un parámetro se envía la dirección de memoria donde se encuentra guardado este parámetro, por lo tanto cualquier modificación en la función destino, se verá reflejada en la función desde donde se invocó la función destino.

1.1.3 Puntero genérico.

¿A que se refiere el termino Puntero Genérico?
Con la definición de puntero que tratamos anteriormente, se puede concluir que un puntero es una variable que guarda una dirección de memoria, sin embargo cuando se mostró la declaración de una variable puntero se observa que hay que especificar el tipo de dato, se sabe que el tamaño de una variable de tipo entero es 2 Bytes, de una de tipo carácter es 1 Byte y de una de tipo flotante es 4 Bytes, entonces surge la pregunta:
¿La dirección de memoria de una variable de tipo flotante es mas grande que la dirección de memoria de una variable de tipo entero o de una de tipo carácter?
La respuesta a esta pregunta es, que no es mas grande, las tres direcciones son del mismo tamaño, es mas, el tamaño de la dirección de cualquier variable es constante, por lo tanto, se define un puntero genérico, una variable que guarda una dirección de memoria sin importar el tipo de información almacenado en esa dirección, cuando declaramos una variable puntero es necesario especificar el tipo de dato para saber como será tratada esta dirección de memoria, para declarar un puntero genérico, basta con colocar que es de tipo void. En caso de que se tenga un puntero de tipo entero y se quiera copiar la información de un puntero genérico o viceversa se deberá hacer uso de un cast (Conversión de tipo).

1.2 cast (Conversión de tipo) en C.

Un cast nos permite hacer una conversión explícita de un tipo de dato a otro, a criterio del programador siempre y cuando estos tipos sean compatibles. Se puede realizar un cast de un puntero de un tipo especifico de dato a un puntero genérico y viceversa.

1.3 Ejercicio.

El siguiente programa en C trata de ilustrar lo que se ha tratado hasta este punto:

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

//Prototipos de funciones
void sumaValor(float *);
void sumaArreglo(void *);

/* Funcion main que recibe parametros de ejecucion */
int main(int argc, char const *argv[])
{
    // Variable de tipo Flotante
    float a = 3.5;

    if(argc == 2)
        a = atof(argv[1]);

    /* El nombre de un arreglo es un puntero
    a su primera direccion de memoria */
    // Arreglo de tipo Entero
    long b[10], i = 0;

    //Declaracion de punteros
    float *ptoa = NULL;
    long *ptob = NULL;

    //Asignacion de punteros
    ptoa = &a;
    ptob = b;

    printf("\n\n TAMANO DE PUNTEROS Y VARIABLES: \n");
    printf("  - Tamanio de la variable(a)    : %d\n", sizeof(a));
    printf("  - Tamanio de la variable(b[0]) : %d\n\n\n", sizeof(b[0]));
    printf("  - Tamanio del puntero (ptoa): %d\n", sizeof(ptoa));
    printf("  - Tamanio del puntero (ptob): %d\n\n\n", sizeof(ptob));
```

```
35      printf("  - Tamanio del *puntero (*ptoa) : %d\n", sizeof(*ptoa));
36      printf("  - Tamanio del *puntero (*ptob) : %d\n", sizeof(*ptob));
37
38
39      //Dando valores al arreglo
40      for (i = 0; i < 10; ++i)
41          b[i] = i;
42      //Valor de la variable llamada a
43      printf("\nValor de (a)     : %.2f\n", a);
44      printf("Direccion de (a) : %p\n", &a);
45
46      //Valor del arreglo llamado b
47      printf("\nValor del Arreglo(b)     :  ");
48      for (i = 0; i < 10; ++i)
49          printf("[%d]",b[i]);
50      printf("\n");
51      printf("Direccion del Arreglo(b) :  %p\n\n", b);
52
53      //llamado a la funcion suma Arreglo
54      //cast de puntero int a puntero void
55      sumaArreglo((void*)b);
56      sumaValor(&a);
57
58      //Valor del arreglo llamado b
59      printf("\nValor del Arreglo(b)     :  ");
60      for (i = 0; i < 10; ++i)
61          printf("[%d]",b[i]);
62      printf("\n");
63      printf("Direccion del Arreglo(b) :  %p\n\n", b);
64
65      //Valor de la variable llamada a
66      printf("Valor de (a)     : %.2f\n", a);
67      printf("Direccion de (a) : %p\n\n", &a);
68
69      return 0;
70  }
71
72  void sumaValor(float * a)
73  {
74    /* Funcion que recibe una referencia a una variable
75    de tipo flotante y modifica su valor sumandole 4.1 */
76
77      *a = *a+4.1;
78      return;
79  }
80
```

```
81 void sumaArreglo(void *pto)
82 {
83   /* Funcion que recibe una referencia a un arreglo
84   de tipo entero y modifica cada una de sus
85   posiciones sumandole 5 al valor ahi guardado */
86
87     int *p = (int*)pto;
88     int i = 0;
89     printf("\nDireccion en Funcion (b): %p\n", p);
90     printf("Valor en Funcion (b[3]) : %d\n\n", (*p+3));
91
92     for (i = 0; i < 10; i++)
93         p[i] = (*(p+i))+5;
94 }
```

1.4 Manejo de memoria dinámica.

Con memoria dinámica, se quiere referir a la memoria que se reserva en tiempo de ejecución. Su principal ventaja frente a la memoria estática, es que su tamaño puede variar en tiempo de ejecución. La biblioteca estándar de C proporciona las funciones calloc, malloc, realloc y free para la gestión de la memoria dinámica, a continuación se describen brevemente:

1. **`malloc`**: Esta función reserva un bloque de memoria y devuelve un puntero genérico al inicio del segmento de memoria. El prototipo de la función es:

   ```
   void *malloc(size_t size);
   ```

   El parámetro size indica el número de bytes a reservar.

2. **`calloc`**: Esta función básicamente realiza el mismo trabajo que la función malloc, con la diferencia que además de reservar la memoria, inicializa a '0' la memoria que reserva. El prototipo de la función es:

   ```
   void *calloc(size_t nmemb, size_t size);
   ```

   El parámetro size indica el número de bytes a reservar por cada elemento.
   El parámetro nmemb indica el número de elementos de tamaño size a reservar.

3. **`realloc`**: Esta función redimensiona la memoria dinámica anteriormente asignada a un puntero, la nueva memoria reservada con esta función al igual que con la función malloc no estará inicializada. El prototipo de la función es:

   ```
   void *realloc(void *ptr, size_t size);
   ```

   El parámetro size indica el número de bytes a reservar (El nuevo tamaño).
   El parámetro ptr es el puntero que referencia el segmento a redimensionar.

4. **free**: Esta función sirve para liberar memoria que se asignó dinámicamente. El prototipo de la función es:

```
void free(void *ptr);
```

El parámetro ptr es el puntero que referencia el segmento a liberar.

1.5 Ejercicio, Manejo de memoria dinámica

Construya un programa en C, el cual reciba como parámetros al ejecutarse el número de filas y el número de columnas de una matriz, luego construya la matriz de forma dinámica, no se puede declarar la matriz utilizando los operadores[ ], es decir no se puede escribir “int matriz[3][3];" se puede utilizar en vez de esto un doble puntero, el valor que poseerá cada campo de la matriz viene dado por la suma de sus índices es decir, en la posición [i][j] estará guardado, el valor i+j, por último se debe mostrar por pantalla la matriz resultante.

2 Acerca de las distribuciones Linux

2.1 API POXIS

POSIX es el acrónimo de Portable Operating System Interface; la X viene de UNIX como seña de identidad de la API. El API POSIX Se trata de un estándar que intenta asegurar la portabilidad entre diferentes sistemas operativos. Dentro del estándar se especifica el comportamiento de las expresiones regulares y de las herramientas más comunes que las usan. Así mismo define un estándar de llamadas al sistema operativo. La librería estándar de C define unas funciones que deben estar en cualquier entorno de desarrollo de C.

2.2 Uso del manual de referencia y comandos básicos

Comandos básicos de la cónsola de comandos (shell) de una distribución linux:

- **ls**, permite listar el contenido de un directorio o fichero.
- **pwd**, imprime nuestra ruta o ubicación al momento de ejecutarlo.
- **mkdir**, crea un directorio nuevo tomando en cuenta la ubicación actual.
- **rm**, permite borrar un archivo o directorio.
- **cp**, copia un archivo o directorio origen a un archivo o directorio destino.
- **mv**, mueve un archivo a una ruta específica, y a diferencia de cp, lo elimina del origen finalizada la operación.

Consulte el manual de referencia que provee el sistema, para abrirlo basta escribir desde su consola:

```
usuario@pc:/$ man man
```