UNIVERSIDAD TECNOLÓGICA DE PANAMÁ
FACULTAD DE INGENIERÍA DE SISTEMAS COMPUTACIONALES
DEPARTAMENTO DE PROGRAMACIÓN DE COMPUTADORAS
MATERIAL GUIADO SOBRE PUNTEROS

PUNTEROS Y ARREGLOS

Los punteros tipo carácter se utilizan con frecuencia para tener acceso a cadenas de caracteres. Cadenas de caracteres y punteros a carácter pueden ser tratados en forma indistinta. El nombre de un arreglo de caracteres sin el uso de un subíndice se considera como puntero al primer elemento del arreglo de caracteres.

La siguiente imagen muestra la declaración de un puntero tipo carácter y su uso potencial:

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

void main(){
    char arr[25], *ptr;
    strcpy(arr, "Bienvenidos al Curso");
    //función strcpy es para copiar una cadena en una variable
    ptr= arr;

    printf("%s \n", ptr);
}
```

```
"C:\Users\giank\Desktop\HPA
Bienvenidos al Curso
```

Al ejecutar el programa se muestra que la variable ptr apunta al inicio de la secuencia de 25 caracteres. Observe que el operador ***&*** no se utilizó para el acceso a la dirección de memoria del primer elemento ya que la variable arr (que es un arreglo), cuando se utiliza sin subíndice, es a la vez el nombre del arreglo y el puntero haría referencia a la dirección del primer elemento del arreglo.

En consecuencia: ***ptr= arr;*** es equivalente a escribir ***ptr= &arr[0];***.

La variable ***ptr*** apunta al primer elemento de la variable, entonces ****ptr*** muestra lo que contiene el primer elemento del arreglo.

Revisemos el siguiente programa:

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

void main(){
    char arr[25], *ptr;
    strcpy(arr, "Bienvenidos al Curso");
    //función strcpy es para copiar una cadena en una variable
    ptr= arr;

    printf("%c \t %c \n", *ptr, arr[0]);
}
```

```
"C:\Users\giank\Desktop\HPA
B       B
```

En la línea del código ***printf("%c \t %c \n", *ptr, arr[0]);*** *al imprimir el contenido de la dirección de memoria almacenada en el puntero podemos darnos cuenta que efectivamente apunta hacia la primera posición del arreglo arr[0].*

Los punteros pueden entonces permitirnos almacenar cadenas de caracteres sin necesidad de requerir un arreglo, veamos el siguiente código fuente:

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

void main(){
    char *ptr;
    ptr= "Bienvenidos al Curso";

    printf("%s\n", ptr);
}
```

```
"C:\Users\giank\Desktop\HP
Bienvenidos al Curso
```

Podemos ver otro ejemplo en el que mediante punteros podemos prescindir de los arreglos de caracteres, ya que con un puntero podemos obtener el mismo resultado. En el siguiente código se presenta la impresión de un arreglo y también podemos observar que se puede realizar el mismo recorrido de cada carácter utilizando un puntero:

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

void main(){
    char arr[30], *ptr;
    int x;

    printf("Introduzca su nombre completo: ");
    gets(arr);

    ptr= arr;

    //Vamos a imprimir el arreglo caracter a caracter
    for(x=0; x < strlen(arr); x++){
        printf("%c ", arr[x]);
    }
    printf("\n\n");

    /*podemos imprimir caracter a caracter el arreglo
    utilizando el puntero*/
    for(x=0; x < strlen(arr); x++){
        printf("%c ", *(ptr+x));
    }
}
```

```
"C:\Users\giank\Desktop\HPA
Introduzca su nombre completo: Pedro Gonzalez
P e d r o   G o n z a l e z

P e d r o   G o n z a l e z
```

Tomando en cuenta que estamos trabajando con el tipo de datos char en el que cada elemento ocupa 1byte, al sumarle 1 a la dirección almacenada en el puntero sería el de la siguiente posición del arreglo. La equivalencia sería la siguiente:

$$*ptr \equiv arr[0]$$
$$*(ptr+1) \equiv arr[1]$$
$$*(ptr+2) \equiv arr[2]$$
$$*(ptr+3) \equiv arr[3]$$
$$*(ptr+N) \equiv arr[N]$$

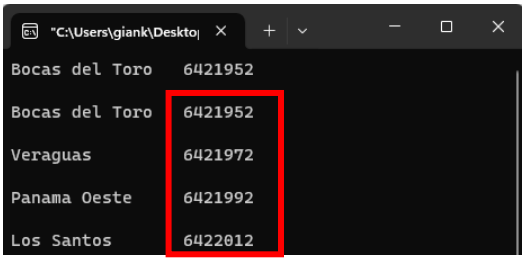
Es más eficiente la declaración de punteros tipo carácter que la declaración de arreglo de caracteres. En el arreglo de caracteres se tiene que definir el tamaño de memoria a utilizar, por ejemplo ***char arr[20];***, esto indica que se tiene que asignar el espacio de 20 bytes aunque no se utilicen todos, en cambio con un puntero tipo char se ocupará únicamente la cantidad requerida de bytes.

ARREGLO DE PUNTEROS

Como ya sabemos, un arreglo es una colección de datos homogéneos, y un puntero es una variable especial que guarda una dirección de memoria. Al ser el puntero una variable, es posible y práctico contar con un arreglo de punteros.

Analicemos el siguiente código fuente:

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

void main(){
    char arr[4][20];
    int i;

    strcpy(arr[0], "Bocas del Toro");
    strcpy(arr[1], "Veraguas");
    strcpy(arr[2], "Panama Oeste");
    strcpy(arr[3], "Los Santos");

    printf("%s \t %u \n\n", arr , arr);
    /*se imprime la cadena de la primera fila y la dirección
     de la primera posición de la primera fila del arreglo*/

    //%u nos permite tener la dirección de memoria en entero

    for(i=0;i<4;i++){
        printf("%s \t %u \n\n", arr[i] , arr[i]);
        /*este ciclo permite imprimir cada fila del arreglo y se
         imprime la dirección de memoria del primer caracter de
         cada fila*/
    }
}
```



Vamos a prestar especial atención a las direcciones de memoria (recuadro rojo de la imagen anterior).

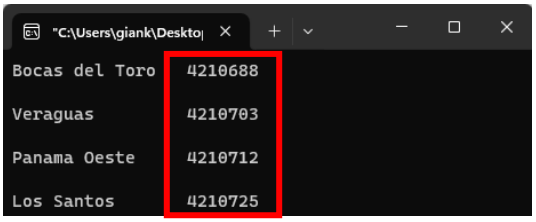
Recordemos que el arreglo declarado es ***char arr[4][20];***, considerando que cada fila corresponde a una cadena de 20 caracteres, entonces el desplazamiento que tenemos en las direcciones corresponde a 20bytes entre cada fila.

Fila	Cadena	Dirección de la primera posición de la fila	Desplazamiento (dir de la fila + 20bytes)
0	Bocas del Toro	6421952	6421952 + 20 = 6421972
1	Veraguas	6421972	6421972 + 20 = 6421992
2	Panama Oeste	6421992	6421992 + 20 = 6422012
3	Los Santos	6422012	6422012 + 20 = 6422032

El arreglo bidimensional arr acepta 3 cadenas con un máximo de 19 caracteres más el carácter de fin de cadena '\0'. Como se puede ver en la ejecución del programa, el compilador reservará espacio para 20 caracteres (incluyendo el fin de cadena) para cada una de las cadenas.

Ahora, observe el siguiente programa:

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

void main(){
    char *arr[4];
    int i;

    arr[0]= "Bocas del Toro"; //15 caracteres incluyendo fin de cadena
    arr[1]= "Veraguas"; //9 caracteres incluyendo fin de cadena
    arr[2]= "Panama Oeste"; //13 caracteres incluyendo fin de cadena
    arr[3]= "Los Santos"; //11 caracteres incluyendo fin de cadena

    for(i=0;i<4;i++){
        printf("%s \t %u \n\n", arr[i] , arr[i]);
        /*este ciclo permite imprimir cada fila del arreglo y se
         imprime la dirección de memoria del primer caracter de
         cada fila*/
    }
}
```



Volvamos a prestar especial atención a las direcciones de memoria (recuadro rojo de la imagen anterior).

En este caso se tiene un arreglo de tres punteros a carácter. En la ejecución solo se guardará espacio para cada una de las cadenas en forma individual, y al ejecutarse el programa se obtendremos lo siguiente:

Fila	Cadena	Longitud de la cadena	Dirección de la primera posición de la fila	Desplazamiento (dir de la fila + longitud de cadena)
0	Bocas del Toro	15 caracteres	4210688	4210688 + 15 = 4210703
1	Veraguas	9 caracteres	4210703	4210703 + 9 = 4210712
2	Panama Oeste	13 caracteres	4210712	4210712 + 13 = 4210725
3	Los Santos	11 caracteres	4210725	4210725 + 11 = 4210736

Como se observó en este caso, existe una diferencia (considerable) entre punteros y arreglos, en particular en el caso de los arreglos de caracteres. Podemos observar que con un arreglo se pierden bytes que no se llegan a utilizar, en cambio con los punteros se optimiza el consumo de memoria ya que solo se ocupa el espacio específico necesario para almacenar cada cadena.

En este caso, cuando sea necesario almacenar en un arreglo varias cadenas, es más eficiente emplear arreglo de puntero.

UNIVERSIDAD TECNOLÓGICA DE PANAMÁ
FACULTAD DE INGENIERÍA DE SISTEMAS COMPUTACIONALES
DEPARTAMENTO DE PROGRAMACIÓN DE COMPUTADORAS
MATERIAL GUIADO SOBRE PUNTEROS

ASIGNACIÓN DINÁMICA DE MEMORIA (PUNTEROS)

Hemos visto cómo el lenguaje C define y utiliza los punteros para acceder a las posiciones de memoria asignadas a un programa.

Otra de las grandes ventajas de la utilización de punteros es la posibilidad de realizar una asignación dinámica de memoria, esto quiere decir, que la reserva de memoria se realiza dinámicamente en tiempo de ejecución, no siendo necesario entonces tener que especificar en la declaración de variables la cantidad de memoria que se va a requerir. La reserva de memoria dinámica añade una gran flexibilidad a los programas porque permite al programador la posibilidad de reservar la cantidad de memoria exacta en el preciso instante en el que se necesite, sin tener que realizar una reserva por exceso en prevención a la que pueda llegar a necesitar.

Es decir, que en C, es posible conseguir nuevas posiciones de memoria que se pueden utilizar en un programa en tiempo de ejecución.

La función malloc (incluida en la librería stdlib.h), permite buscar un espacio de memoria libre del tamaño deseado y separarlo, de modo que la dirección del primer byte pueda ser asignada a un puntero.

Formato: void * malloc (tamaño);

En este formato, tamaño indica la cantidad de bytes que se desea tenga el bloque de memoria. El tipo de dato void * indica que la dirección que devuelva la función no está relacionada con ningún tipo de dato, por esta razón cuando se emplee esta función, se debe aplicar una operción cast al resultado devuelto por malloc para asociarlo a un tipo de dato específico.

Si por alguna razón, no se logra ubicar un espacio de memoria del tamaño deseado o el tamaño solicitado es cero, la función malloc devuelve NULL.

Ejemplo:

```c
#include<stdio.h>
#include<stdlib.h>  //librería para trabajar con función malloc

void main(){
    int *ptr; //declaración de puntero que referencia a un espacio para una variable int

    ptr= (int *) malloc(sizeof(int)); //se obtiene una dirección de memoria libre con el espacio para un dato tipo int y se le asigna a ptr
    /*DETALLE DE LA ASIGNACIÓN DE MEMORIA
    (int *) define que la dirección de memoria debe ser para un dato tipo int
    sizeof(int) devuelve el tamaño en bytes de una variable de tipo int
    malloc(sizeof(int)) obtiene una posición de memoria libre con el tamaño de una variable tipo int
    ptr= se asigna la dirección de memoria obtenida de forma dinámica al puntero ptr
    */
    scanf("%i", ptr); //se realiza la lectura de un dato tipo int y se almacena en la dirección de memoria guardada en el puntero ptr

    printf("El valor ingresado es: %i", *ptr); //se imprime el valor guardado en la dirección de memoria que posee ptr
}
```

Si quisiéramos obtener un espacio en memoria para un dato de tipo float el código sería el siguiente:

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

void main(){
    float *ptrF;

    ptrF= (float *) malloc(sizeof(float));
    scanf("%f", ptrF);

    printf("El valor ingresado es: %f", *ptrF);
}
```

Podemos notar que hemos almacenado un dato de tipo float sin la necesidad de haber declarado una variable, todo fue realizado mediante punteros a los que se le asignó un espacio en memoria de manera dinámica durante la ejecución del programa.

Por otro lado, si a la expresión sizeof(int) se le multiplica por un valor entero, se habría definido un espacio de memoria denominado "arreglo dinámico", esto quiere decir que el puntero podrá se manejado como un arreglo común, con la diferencia que su tamaño se definió en tiempo de ejecución y no en tiempo de compilación como se define en los arreglos comunes.

Ejemplo:

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

void main (){
    int *ptr, i, tam;

    printf("Ingrese el número de datos del arreglo: ");
    scanf("%d", &tam);

    ptr= (int *) malloc(tam * sizeof(int)); //se asigna a ptr la dirección de un bloque de memoria equivalente a un arreglo de tipo int de tamaño tam

    //ALMACENAR LOS DATOS EN LAS DISTINTAS DIRECCIONES DE MEMORIA DE PTR COMO SI FUESE UN ARREGLO
    printf("\nIntroduzca los %i numeros: ", tam);
    for(i=0; i< tam; i++){
        scanf("%i", (ptr+i)); //(ptr+i) desplaza las direcciones de memoria como si se tratara de un arreglo de i posiciones
    }

    //IMPRESIÓN DE LOS DATOS ALMACENADOS EN LAS DIRECCIONES DE MEMORIA DEL PUNTERO PTR COMO SI FUERA UN ARREGLO
    for(i=0; i< tam; i++){
        printf("%i \t", *(ptr+i));
    }
}
```

Existe una función para liberar la dirección de memoria asignada de manera dinámica a un puntero, esta función es free().

Formato: free(variable puntero a la que se le asignó de forma dinámica una dirección de memoria disponible);

Para el caso del ejemplo anterior en el que se asignaba un bloque de direcciones de memoria del tamaño de un arreglo de tipo int de tamaño definido por el usuario, quedaría de la siguiente manera para liberar todo el bloque de memoria asignado dinámicamente.

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

void main (){
    int *ptr, i, tam;

    printf("Ingrese el número de datos del arreglo: ");
    scanf("%d", &tam);

    ptr= (int *) malloc(tam * sizeof(int)); //se asigna a ptr la dirección de un bloque de memoria equivalente a un arreglo de tipo int de tamaño tam

    //ALMACENAR LOS DATOS EN LAS DISTINTAS DIRECCIONES DE MEMORIA DE PTR COMO SI FUESE UN ARREGLO
    printf("\nIntroduzca los %i numeros: ", tam);
    for(i=0; i< tam; i++){
        scanf("%i", (ptr+i)); //(ptr+i) desplaza las direcciones de memoria como si se tratara de un arreglo de i posiciones
    }

    //IMPRESIÓN DE LOS DATOS ALMACENADOS EN LAS DIRECCIONES DE MEMORIA DEL PUNTERO PTR COMO SI FUERA UN ARREGLO
    for(i=0; i< tam; i++){
        printf("%i \t", *(ptr+i));
    }

    free(ptr); //de esta manera se libera todo el espacio en memoria asignado a ptr (arreglo dinámico)
}
```