INSTITUTO POLITÉCNICO NACIONAL
UNIDAD PROFESIONAL INTERDISCIPLINARIA EN INGENIERÍA Y TECNOLOGÍAS AVANZADAS

Práctica No. 6 .Uso de apuntadores.

Unidad Temática: IV Manejo de Apuntadores y Estructuras.

Lugar de realización: Laboratorio de Cómputo **Duración:** 9.5 hrs.

Objetivo

Desarrollo de programas utilizando los apuntadores para el paso de parámetros por valor y por referencia a funciones.

Resultados Esperados

- Comprensión del uso de apuntadores y variables
- Manipulación de memoria dinámica
- Uso del direccionamiento de un apuntador a arreglos
- Comprensión y manejo de parámetros por valor y por referencia en funciones

Pre-reporte.

Para el buen desarrollo de la práctica el alumno entregará un trabajo previo que responda las siguientes preguntas:

1. ¿Qué es un apuntador?
2. ¿Para qué pueden servir los apuntadores? (mínimo 5)
3. ¿Cuál es la diferencia entre paso por valor y paso por referencia de funciones?
4. ¿Para qué sirve el operador *ampersand* (`&`)?
5. ¿Para qué sirve el operador *asterisco* (`*`) en apuntadores?

Nota: El pre-reporte cuenta como calificación de la práctica.

"Apunta a la luna. Incluso si fallas aterrizarás entre las estrellas"
(Les Brownn)

Material y Equipo.

- IDE Dev-Cpp, CodeBlocks o compatible.
- Computadora

Introducción.

Memoria

La memoria se divide en cuatro

- **Segmento de Código:** Guarda las instrucciones en lenguaje máquina de nuestro programa
- **Segmento de Datos:** Guarda las variables globales del programa y las librerías que ocupamos.
- **Segmento de Pila (Stack):** Contiene los llamados a función apilados, sus parámetros, y variables locales. El main al ser una función se guarda ahí.
- **Segmento Heap:** Es el único fragmento cuya asignación es dinámica, es decir no se asigna al momento de la compilación, aquí se guardan los elementos creados dinámicamente en la ejecución del programa.

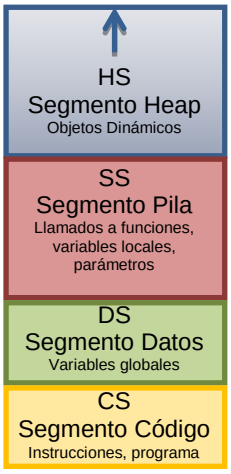


Figura 6.1. Segmentos de Memoria

La **memoria** se compone de posiciones de memoria numeradas (llamadas bytes). En el siguiente fragmento de código se declara una variable `x` como un entero (`int`) cuyo tamaño son 4 bytes por lo que ocupa un fragmento de memoria de 4 posiciones de memoria adyacentes que van de la posición `0XFF04` a la `0XFF07`, la dirección de la primera de esas cuatro es la que se toma en cuenta como dirección de esa variable de tal forma que la dirección de x es `0XFF04`.

```
main(){
     int x;
     x=8;
}
```

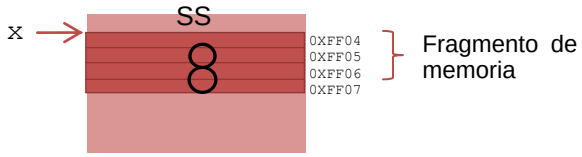


Figura 6.2. Fragmento de memoria

Apuntador

Un **apuntador** es la dirección en memoria de una variable, es decir la dirección que se usa para nombrar dicha variable, se llama apuntador porque podemos pensar que la dirección apunta a la variable. Un apuntador se puede guardar en una variable, por ejemplo `p`, cuyo tipo de dato debe de ser del tipo del dato que guarda, por ejemplo si va a guardar un número entonces `p` debe ser del tipo `int` o `float`. Para señalar que es un apuntador se usa un asterisco `*`.

Declaración:

Tipo _dato * nombre_variable ;

```
int *p;
```

Se usa el operador & para determinar la dirección de una variable, por ejemplo `p` contiene la dirección de `var`.

```
int var;
int *p;
p=&var;
```

Para el manejo de la memoria dinámica en **C** usamos `malloc`, `calloc`, `realloc` y `free`; mientras que en **C++** se usa `new` y `delete`. Malloc nos pemite reservar la cantidad de bytes especificada y nos regresa un apuntador genérico (void) y se encuentra en la librería `stdlib.h`

Prototipo:

Apuntador_genérico malloc (número_bytes);

```
void* malloc ( unsigned int);
```

En el siguiente ejemplo podemos observar el uso de la memoria Heap, la cual es expandible.

```
main(){
        int *ptr;
    ptr = (int*)malloc(sizeof(int));
    *ptr = 8;
}
```

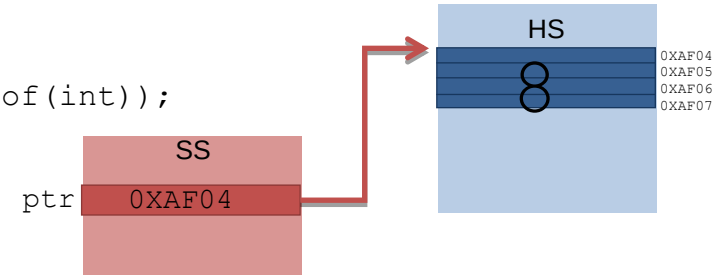


Ilustración 6.3 Reservación de memoria

Otra forma de hacerlo es declarar la variable y después pasar la dirección de memoria de esa variable a una apuntador con el operados &.

```
int variable;
int * apuntador = &variable
*apuntador = 20; //Asignamos un valor al contenido en esa posición de memoria.
```

Paso por Valor

Una variable solo existe en la función en la que se crea y nada más, a través de los parámetros podemos pasar el valor de una determinada variable a otra función, pero lo que pasamos es el valor y no la variable en sí.

Paso por Referencia

Hay veces que nos interesa que una función modifique una variable que no pertenece a ella, es decir, fuera de su ámbito; lo cual se hace a través de punteros (o referencias). La idea es que como solo se puede pasar el valor de una variable a una función lo que hacemos es pasar la dirección de una variable la cual puede

ser reservada con malloc o a través de del operador de indirección & con el que podemos acceder al contenido de la variable original.

Desarrollo.

Ejemplo:

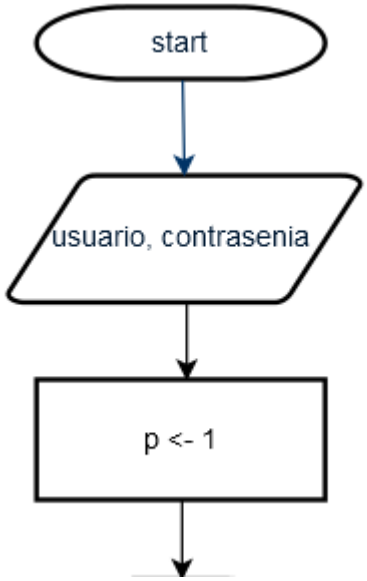
Simulación de usuario y contraseña. Escriba un programa que permita ingresar el nombre de un usuario y su contraseña la cual estará compuesta de un carácter y una secuencia de números. Una vez que se solicita por primera vez, el código no deberá acceder a las variables de manera directa (use apuntadores), el usuario podrá acceder a un menú en el que puede ver su usuario y contraseña o validarla.

Solución:

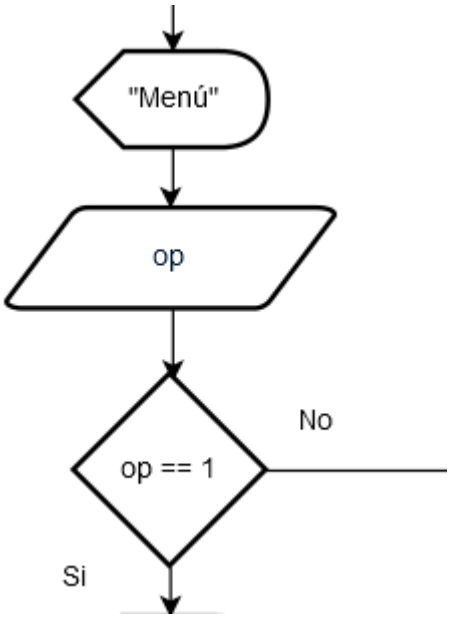
Diagrama de Flujo

La lógica del programa en general la podemos observar en el siguiente diagrama de flujo, aunque el manejo de memoria se ve reflejado hasta el programa en C.

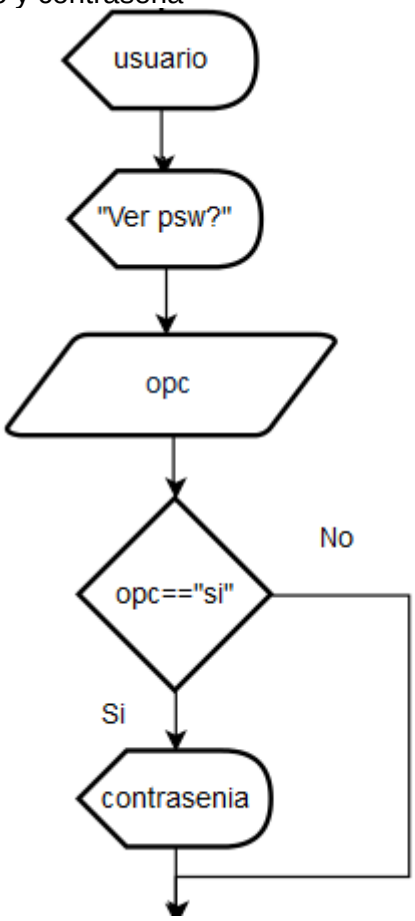
1. Inicialización de variables



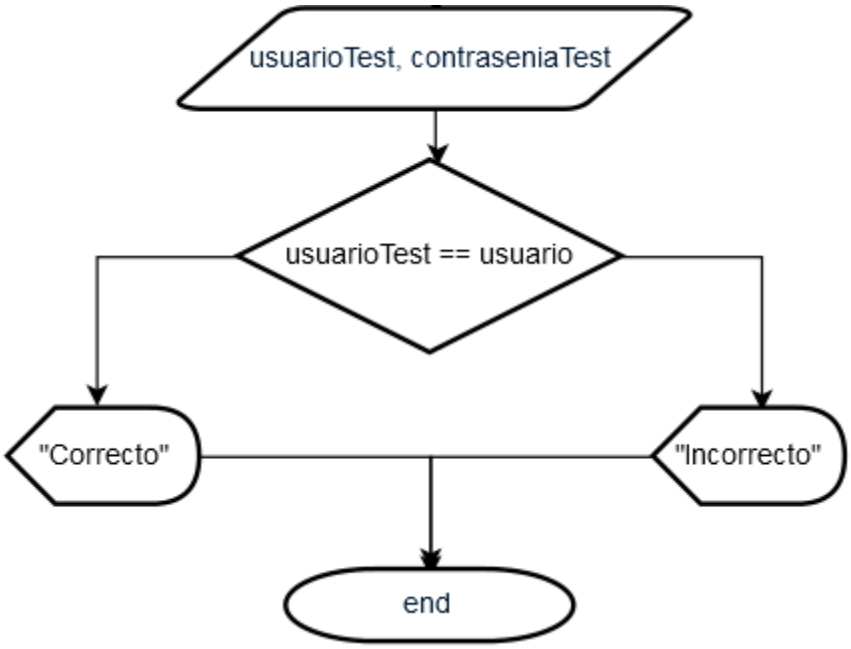
2. Despliegue de opciones



3. Opción imprimir usuario y contraseña



4. Opción validar usuario y contraseña



Código *Ver código 6contraseniaDir.c anexo en disco*

1. Declaración de variables

```
#include <stdio.h>
int main(){
char usuario[10]; //usuario
char uTest[10]; //para validación de usuario
char caracter1, cTest; //character de la contraseña
int numero, nTest; //entero de la contraseña
int op; //para seleccionar opciones
char op2; //para seleccionar opciones
```

2. Declaración de apuntadores y asignación de dirección de memoria

```
char *ap_caracter=&caracter1;
int *ap_num=&numero;
```

3. Ingresar usuario y contraseña por primera vez

```
printf("Ingrese su usuario: ");
scanf("%s",&usuario);
printf("Ingrese su contrasenia (1caracteres y número):");
fflush(stdin);
scanf("%c%d",&caracter1,&numero);
```

4. Menú con opciones y Ciclo para repetir los pasos 4 al 6

```
while(op==1||op==2){
   printf("Seleccione:\n");
   printf("1. Ver usuario y contrasenia\n");
   printf("2. Validar usuario y contrasenia\n");
   fflush(stdin);
   scanf("%d",&op);
```

```
        if(op==1){
            //acciones para ver
        }else if(op==2){
            //acciones para validar
        }
    }
```

5. Ver usuario y contraseña

```
printf("Usuario: %s\n",usuario);
printf("Contraseña: \n");
printf("  entraría a la dirección %p y %p, está seguro de
       continuar (S/N)) ",ap_caracter,ap_num);
fflush(stdin);scanf("%c",&op2);
if(op2=='S')
   printf("contrasenia... %c%d\n",*ap_caracter,*ap_num);
```

6. Verificar usuario y contraseña

```
printf("Ingrese su usuario: ");
scanf("%s",&uTest);
printf("Ingrese su contrasenia (3 caracteres y un
       número):");
fflush(stdin);
scanf("%c%d",&cTest,&nTest);
if(strcmp(uTest,usuario)==0){
    printf("CORRECTO!!\n");
}else{
    printf("INCORRECTO!!\n");
}
```

Cuestionario

Resuelva los siguientes cuestionamiento:

1. Describa línea por línea el código de paso por valor siguiente:

Código paso por valor	
`#include <stdio.h>`	Se incluye la libreria estandar
`int sumar(int a,int b);`	Prototipo de la funcion sumar
`main (){`	Inicio funcion main
`int num1;`	Declara num1 variable entera
`int num2;`	Declara num2 variable entera
`num1=5;`	Asigna el valor 5 a num 1
`num2=8;`	Asigna el valor 8 a num 2
`int total=sumar(num1,num2);`	Llama a la funcion sumar
`}`	
`int sumar(int a, int b){`	Declaracion de funcion sumar
`int c=0;`	Declara a c como entero igual a 0
`c=(a)+(b);`	Suma los argumentos de la funcion
`return c;`	Regresa el resultado de la suma
`}`	

2. Describa línea por línea el código de paso por referencia siguiente:

Código paso por referencia	
`#include <stdio.h>`	Incluye librerias estandar
`#include<stdlib.h>`	
`int sumar(int* a,int* b);`	Prototipo de la funcion sumar
`main (){`	Inicia funcion main
`int* num1=(int*)malloc(sizeof(int));`	Reserva espacio de memoria del tamaño de un entero
`int* num2=(int*)malloc(sizeof(int));`	
`*num1=5;`	Inicializacionde los apuntadores
`*num2=8;`	
`int total=sumar(num1,num2);`	Llamado a la funcion sumar
`}`	
`int sumar(int* a, int* b){`	Declaracion de funcion sumar
`int c=0;`	Declara c=0
`c=(*a)+(*b);`	Realiza la suma de los apuntadores
`return total;`	Devuelve el resultado de la suma
`}`	

3. Programe los códigos del punto 1 y 2 e imprima en ambos casos el `total`, `num1` y `num2`. Posteriormente responda las siguientes preguntas:
   a. En el programa de paso por valor, ¿Qué sucede si modifica el valor de la variable `a` en la función sumar con respecto al valor de `num1`?

      Se mantiene el mismo valor

   b. En el programa de paso por valor, ¿Qué sucede si modifica el valor de la variable `total` después del llamado de función con respecto al valor de `c` en la función sumar?

      Se modifica el valor del total pero se mantiene el valor de c

   c. En el programa de paso por referencia, ¿Qué sucede si al inicio de la función sumar introduce la línea de código `a=4;`?

      El programa marca un error al compilar

   d. En el programa de paso por referencia, ¿Qué sucede si al inicio de la función sumar introduce la línea de código `*a=4;`?

Se modifica el valor del total

e. Calcule el uso de memoria para ambos programas y exponga la diferencia.

El de paso por referencia pesa 11 bytes mas que el de paso por valor

Programas

Al pasar un arreglo a una función, el paso se hace por referencia. Escriba un programa que:

a. Contenga dos arreglos en el `main` de 5 elementos enteros. Por ejemplo:
```
int arreglo1 [ ] = {2, 7, 1, 9, 3};
```
b. Agregar las siguientes funciones; las cuales podrán ser seleccionadas en un menú:
   i. Una función que imprima un arreglo
      Argumentos de Función:
      Arreglo de enteros
      Procedimiento:
      1) **Iniciar i=0**
      2) **Obtener elemento i del arreglo**
      3) **Imprimir el elemento del arreglo**
      4) **Si i es menor al tamaño del arreglo regresar a 2**
      Retorno de Función:
      Ninguno
   ii. Una función que reciba dos arreglos y “regresar” un tercero con la suma de ambos. (imprimir resultado desde main)
      Argumentos de Función:
      3 arreglos de enteros
      Procedimiento:
      1) **Iniciar i=0**
      2) **Obtener elemento i del arreglo1 y ese mismo de arreglo2**
      3) **Sumar esos elementos y guardarlos en el arreglo3**
      4) **Si i es menor al tamaño del arreglo regresar a 2**
      Retorno de Función:
      Ninguno de manera explícita
   iii. Que “regrese” el arreglo ordenado de manera invertida pero sin modificar el original
      Argumentos de Función:
      2 arreglo de enteros
      Procedimiento:
      1) **Copiar el arregloOriginal en un segundo arreglo**
      2) **Ordenar el arreglo Copia**
      Retorno de Función:
      Ninguno de manera explícita
   iv. Una función que reciba un arreglo de números y “regrese” un arreglo con las letras ‘P’ o ‘I’ dependiendo si el número es par o impar (imprimir resultado en main)
      Argumentos de Función:
      Un arreglo de enteros y un arreglo de caracteres
      Procedimiento:
      1) **Para cada elemento del arreglo**

2) **Obtener el módulo**
3) **Si el módulo es 0 escribir una 'P' en esa posición en el arreglo de caracteres**
4) **Si no, escribir una 'I' en la misma posición en el arreglo de caracteres**

Retorno de Función:
Ninguno de manera explícita

Proyectos (opcionales).

Hacer el conjunto de funciones que permitan la **manipulación de cadenas** de caracteres recibidas como apuntadores, las funciones incluyen: copiar, obtener tamaño, cambiar a mayúsculas, obtener la posición en la que se encuentra determinada letra y decir si la cadena contiene determinadas subcadena. Guardar como una librería.

Realizar la administración de un **autobús de pasajeros** donde los asientos son representados en una matriz de 4x20 (20 filas del autobús y 4 asientos por fila) y hacer una función que permita reservar un lugar, cancelar algún viaje liberando el espacio y cancelar el camión liberando todo el autobús.

Ponderación de la Práctica.

Sección	Elemento a Evaluar
Pre-reporte	Elementos básicos de apuntadores y funciones
Ejercicio 1	Paso por valor
Ejercicio 2	Paso por referencia y reservación de memoria
Ejercicio 3	Manipulación de memoria dinámica, pasos por valor y referencia
Programa Corto	Apuntadores en funciones, paso por referencia
Problema	

Bibliografía.

Deitel P.J.y Deitel H. M., Como Programar C++, Ed. Prentice Hall, 6 ª Impresión, México, 2009, ISBN: 970-26-1273-X, Págs: 1-1050.

Deitel P.J. y Deitel H. M., Como programar en C#, Ed. Prentice Hall, 2ª Impresión, México, 2007,ISBN: 9702610567, Págs: 1-1080.

Guardati, Silvia, Estructura de Datos Orientado a Objetos con C ++, Ed.Prentice, México 2007, ISBN: 9702607922, Págs: 1–183.

NOTA: Presentar el reporte en un documento (PDF o DOC), el código fuente y la impresión de la pantalla de ejecución.

"Puedes apoyarte de tus compañeros y profesor para aclarar tus dudas"