**Trabajo Práctico Nº 9.1**

1. Todos los datos se almacenan en memoria, dependiendo de el tipo de dato es la cantidad de espacios de memoria que va a requerir para almacenarlo, cada espacio es un byte y los diferentes 1 y 0 conforman la información que le da forma al dato. Por ejemplo si usamos un dato de tipo int = 5 se vería algo asi si pudiéramos ver la memoria.  
   00000000 00000000 00000000 00000101 🡪 5 (int 🡪 4 espacios)
2. Un puntero, como su nombre lo dice, apunta a un espacio especifico de la memoria. A través de el podemos acceder a la dirección de memoria que esta ubicado el dato que queremos, el puntero es del tipo de dato al que esta apuntando, si estamos apuntando a la dirección de memoria de un int el puntero también es del mismo tipo, no por que contengan el mismo tipo de dato, si no para saber cuantos espacios de memoria tiene que tomar ese puntero.
3. El operador & devuelve la dirección de memoria de un dato y se puede usar para asignarle a un puntero existente la dirección de memoria de un dato, por ejemplo si yo tengo un dato int i = 5 puedo mostrar por pantalla la dirección de memoria de i con un cout o también puedo asignársela a un puntero del tipo int.  
   En cambio el operador \* accede a lo que hay en la dirección de memoria, ósea que si yo le asigne a un puntero la dirección de memoria de i con &i entonces cuando yo quiera mostrar en pantalla \*puntero, lo que voy a mostrar no es la dirección de memoria, si no el valor almacenado(en este caso 5).
4. El puntero tiene que tener una dirección valida asignada, se le asigna una desconocida si no lo hacemos cuando es creado y solo tiene que ser utilizado cuando se le asigna una. Un puntero puede tener una dirección valida asignando la dirección de otro dato con el operador &, asignando otro puntero del mismo tipo de dato o asignándole el valor NULL que es nulo ósea que no apunta a nada.
5. El problema de un puntero no inicializado es que se le asigna una dirección de memoria al azar, no conocemos esa dirección y tampoco sabemos que puede contener. Si creamos un puntero y tratamos de modificar su valor SIN ANTES asignarle una dirección validad quien sabe que dato estamos modificando? Podemos estar cambiando código del mismo programa o hasta del sistema operativo, puede traer problemas muy grabes.
6. Es seguro por que lo podemos usar para detectar errores. Si tenemos un puntero NULL y tratamos de darle un valor va a dar un error de ejecución por que tratamos de acceder a un dato a través de un puntero nulo que si bien es malo, podemos saber donde esta el error y donde tenemos que buscar y trabajar para solucionar el problema.
7. Cuando nosotros copiamos un puntero no estamos solo copiando el mismo valor dos veces, no tenemos dos 5 nada mas, tenemos también mas puntos de acceso al mismo dato y también podemos usar las comparaciones. Cuando nosotros comparamos punteros estamos comparando las direcciones entre ellos(tienen que ser del mismo tipo de datos los punteros comparados) y podemos usarlo para poner condiciones en nuestro código, por ejemplo si puntero1 y puntero2 apuntan a la misma dirección ejecutamos una parte de código y si no ejecutamos esta otra parte.
8. Usando el typdef podemos crear tipos de punteros con un tipo base como por ejemplo  
   *typedef double \*tDoublePtr* y podemos usarlos como cualquier otro tipo de dato y trabajarlos de la misma forma.
9. Como podemos usar punteros como tipo de dato también podemos usarlos como estructuras, como cualquier registro. En este caso vamos a tener que usar el operador -> para utilizar un tipo de dato que este almacenado dentro.  
   Si tenemos una estructura con códigos y nombres, le asignamos su dirección a un puntero, después para utilizarlo podemos usar puntero - > código para usar el código de esa estructura en nuestras funciones.
10. Un puntero a constante es un puntero que le asignamos la dirección en la que esta una constante, ósea que no podemos modificar esa constante pero podemos volver a utilizar el puntero para asignarle otra dirección de memoria u otro puntero .  
    En cambio un puntero constante es un puntero que no podemos modificar su dirección de memoria, una vez asignado esta no puede volver a modificarse pero podemos trabajar con el dato que contiene y usarlo para lo que necesitemos.  
    La diferencia entre ambos se puede ver en como se escriben cuando los creamos:  
    const tipo \*puntero; Puntero a una constante

tipo \*const puntero; Puntero constante

1. En el lenguaje de C no existe un mecanismo para pasar por referencia, solo se puede pasar un parámetro por valor, entonces que hacemos? Usamos a los puntero para simular un paso por referencia.  
   void incrementa(int \*punt)  
   {  
   (\*punt)++;  
   }  
   int entero = 5;  
   incrementa(&entero);  
   cout << entero << endl;  
   (muestra 6 en pantalla)  
     
   Con este ejemplo puntero recibe la dirección de memoria en la que tenemos guardado el 5 y después la modificamos.
2. La relación entre los punteros y los arrays es tan estrecha que pueden ser usados de forma casi idéntica, una variable de tipo array puede considerarse como un puntero de tipo arreglo.  
   Ejemplo:   
   int números[10] =   
   {1, 3, 4, 2, 7, 9, 6, 8, 5, 0};  
   cout << \*números << endl;  
     
   Me va a mostrar 1 en pantalla ese código, siempre va apuntar al primer elemento de un array pero se puede cambiar eso con diferentes herramientas, el nombre de un array es un puntero constante por defecto
3.   
   Usando este grafico para representar como el sistema ve la memoria vamos a explicar principalmente 3 partes de el.  
   *Datos globales*: Son los datos del programa que son declarados fuera de todos los subprogramas y cualquier parte del código puede utilizar según lo necesite.  
   *Pila*: En la pila se guardan todos los parámetros por valor y las variables locales que forman los subprogramas.  
   *Monton*: Se guardan todos los datos dinámicos que se crean y se destruyen a medida que se necesita, el sistema de gestión de memoria dinamica(SGMD) reserva espacio cuando una variable lo necesita para almacenar información y una vez utilizado se libera para poder contar de nuevo con esa memoria.
4. Si pensamos el programa usando datos dinámicos estamos usando datos que se crean y se destruyen a necesidad para no ocupar mucho espacio sin necesidad, no tenemos un limite máximo ya que una vez utilizado se destruye y se libera el espacio.  
   El sistema va a ajustar el uso de memoria según la necesidad entonces no vamos a sobrecargar la memoria principal y con la tecnología que tenemos ahora podemos aprovechar este tipo de tecnología mucho mas ya que no estamos tan limitados, pero igual tenemos que saber como utilizar la memoria dinámica o podemos causar problemas serios si no manejamos bien cuando utilizar o liberar el dato.

**Marco Practico:**

**1)**

int x = 5, y = 12, z;

int \*p1, \*p2, \*p3;

p1 = &x;

p2 = &y;

z = \*p1 \* \*p2;

p3 = &z;

(\*p3)++;

p1 = p3;

cout << \*p1 << " " << \*p2 << " " << \*p3;//va a devolver 61 12 61, ya que antes de mostrar por pantalla,igualamos a p1 con p3

//van a devolver el mismo resultado.

**2)**

int dato = 5;

int \*p1, p2; //el int p2 no esta declarado como puntero de tipo int ya que no tiene el \*.

p1 = &dato;

p2 = p1;

cout << \*p2;

**3)**

double d = 5.4, e = 1.2, f = 0.9;

double \*p1, \*p2, \*p3;

p1 = &d;

(\*p1) = (\*p1) + 3;

p2 = &e;

(\*p3) = (\*p1) + (\*p2);

cout << \*p1 << " " << \*p2 << " " << \*p3; //El problema es que a p3 no le estamos asignando ningun espacio de memoria

//y no sabemos que dato estamos remplazando cuando le asignamos un valor,la forma correcta seria crear un int sin valor

// y asignarle un espacio de memoria para p3 y despues usarlo.

**4)**

const int m = 8;

const int \*const p = &m;

cout << \*p; //Declaramos un int constante igual a 8 y despues creamos un puntero del tipo constante int y que ademas sea un puntero constante

//asi no pueden ser cambiados ni a quien apunta el puntero ni el valor que contiene.

5)

typedef struct {

string nombre;

double sueldo;

int edad;

} tRegistro;

void funcion(tRegistro \*reg, double \*irpf, int \*edad);

int main()

{

return 0;

}

void funcion(tRegistro \*reg, double \*irpf, int \*edad)

{

    const double tipo = 0.18;

    double a = 0;

    int b = 0;

    reg.edad++;

    irpf = &a;

    edad = &b;

    \*irpf = reg.sueldo \* tipo;

    \*edad = reg.edad;

}

//Modificamos la funcion para que cree punteros que iguale a los valores que necesitamos tomar en la funcion

// y despues los asignamos a un espacio de memoria,despues es bastante similar como sigue la funcion.