**Almacenando una dirección en un puntero**

* Cada variable tiene una dirección.
* Aun cuando no conozcamos la dirección de una variable, podemos almacenar esa dirección en un puntero.
* Supongamos que tenemos una variable entera llamada edad. Ära declarar un puntero que almacene la dirección de la variable edad escribimos la siguiente sentencia:

**Int \*pEdad = NULL;**

NULL es una constante declarada por el sistema estándar que vale 0, que apunta a ninguna dirección de memoria. (se escribe mayúscula y por eso es constante).

**Declaración de un puntero**

* Cuando declaramos un puntero es conveniente que este apunte inmediatamente a alguna dirección.



* En el ejemplo anterior lo inicializamos a NULL.
* NULL es una constante cuyo valor es 0.
* Un puntero inicializado a NULL se llama puntero nulo.

**Nullptr**

* La mayoría de los compiladores nuevos tienen una constante especial llamada nullptr que representa un puntero nulo, por lo que es recomendable utilizarla en vez de NULL.

**Int \*pEdad = nullptr;**

**Asignación del puntero**

**int edad = 25; //declaración de la variable  
int \*pEdad = nullptr; //Declara el puntero  
pEdad = &edad; //Pone la dirección de edad en pEdad**

**Operador de induración**

* El operador de indireccion es el \*
* Se le conoce también como operador de desreferenciacion.
* Cuando un puntero es desreferenciado, se obtiene el valor almacenado en la dirección a la que apunta en puntero.

Se obtiene el valor que hay almacenado en ese lugar

**Indireccion**

* El puntero provee acceso indirecto al valor de la variable cuya direccion el almacena.
* Para asignar el valor de la ariable sueldo a una nueva variable otroSueldo por medio del punto pSueldo. Se haria lo siguiente:

float sueldo = 25654.34; //Crea la variable dueldo  
float \*pSueldo = &sueldo; //pSueldo apunta a la direccion de sueldo  
  
float otroSueldo; //Crea otra variable  
otroSueldo = \*pSueldo; //Asigna el valor en pSueldo (25654.34) a otroSueldo

* *(\*)Aterisco me trae el valor de la direccion del puntero*

*Da lo que tiene el valor de la variable a donde el apunta.*

* *(&) dame la direccion de la variable que lleve & delante*

*ejemplo:*

*Int edad = 20;*

*Cout <<&edad <<endl;*

*Imprime la direccion donde esta ese valor. Ejemplo*

**Confusiones**

* Es muy importante distinguir entre:
* Un puntero
* La direccion que el puntero almacena.
* El valor almacenado en la direccion a la que apunta el puntero.
* Ya que no tener este bien claro es la fuente de la mayoria de las confusiones con los punteros.

**Manipulación de datos con puntero.**

* Una vez que se asigna la dirección de una variable a un puntero, podemos utilizar el puntero para acceder a los datos que se encuentran en la variable.
* El siguiente programa muestra como se utilizan los punteros para manipular los valores de las variables.

**Contenido de los puntero**

* Los punteros permiten manipular direccion sin siquiera conocer los valores almacenados en ellas.
* El siguiente programa muestra el contenido de los punteros.

**Para que se utilizan los punteros**

* Manejar datos en los monticulos (heap)
* Acceder a datos y funciones muembro de las clases.
* Para pasar variables por refrencia a las funciones.

**Ejercicio**

* Modifique el programa RevisaPuntero.cpp paa que multiplique tuEdad y \*pEdad y almacene el resultado en una nueva variable. Desplique esa variable. Piense como el compilador puede notar la diferencia entre operador \* para la multiplicacion y el operaro \* ara desreferenciar pEdad.
* Modifique a RevisaPuntero.cpp para usar el puntero desreferenciado \*pEdad para cabiar el conenido de miEdad o de tuEdad.

Monticulos (heap)

El heap es el nombre que se le da a un bloque de memoria RAM utilizada para almacenar variables dinamicas.

Las variables locales se almacenan en la pila (stack), pero cuando la funcion rtorna, son removidas de la pila.

La ventaja del heap es que la memoria reservada se mentiene disponible hasta que uno,explicitamente la libere.

**Sentencia new**

* Para reversar (allocate) memoria en el heap utlizamos la sentencia new seguida por el tipo de objeto que queremos reservar, de forma que el compilador sepa cuanta memoria es requerida para ese tipo de objeto.
* El valor retornado por la sentencia new es una direccion de memoria, por lo que debe ser asignada a un puntero.

Unsigne shot int \*pPuntero;  
pPuntero = new unsigned short int;

**Asignar un valor en el heap**

* Una vez creado el puntero y reservada la memoria, podemos almacenar valores en ese lugar de la manera que yo hemos visto.

\*pPuntero = 50;

*Por ejemplo desrefenciando le asigno lo que yo quiera*

**Sentencia delete**

* Para liberar la memoria utlizada en el heap, utilizmos la sentencia delete.
* Para utilizar la sentencia delete escribimos:

Delete pPuntero;

* Cuando usamos delete estamos liberando la memoria a la que apunta el puntero

*Es recomendable despues que se borrar asignaler nullptr*

* Cuando usamos delete en un puntero la memoria la que apunta es liberada. Si volvemos a usar delete sobre ese puntero, el programa se abortara.
* Para evitar esto cuando liberemos un puntero, debemos asignarle nullptr.
* Si usamos delete sobre un puntero que apunta a nullptr no hay peligro de que el programa aborte

Delete pPuntero;  
pPuntero = nullprt;

Punteros avanzados

Creacion de objetos en el heap

De la misma forma que podemos crear un puntero a un entero, a un char, o a un float, tambien podemos crear punteros a cualquier tipo de objetos.

Si declaramos un objeto de tipo Gato, podemos declarar un puntero a la clase e instanciar un objeto Gato en el heap, tal como lo hacemos en la pila.

*Gato \*pGato = new Gato;*

Esto llama al constructor por defecto.

Borrando objetos

Cuando borramos un puntero a un objeto en el heap, el destructor del objeto es llamado antes de que se libere la memoria. Esto le permite a la clase limpiar todo, tal como lo hace con los objetos destruidos en la pila.

CreaHeap.cpp

class GatoSimple

{

public:

GatoSimple();

~GatoSimple();

private:

int edad;

};

GatoSimple::GatoSimple()

{

cout << "Se llamo el constructor" << endl;

edad = 1;

}

GatoSimple::~GatoSimple()

{

cout << "Se llamo el destructor" << endl;

}

int main()

{

cout << "GatoSimple Misulino ..." << endl;

GatoSimple \*pMiso = new GatoSimple;

cout << "Borrado de pMiso ..." << endl;

delete pMiso;

cout << "Fin, adios Misulindo ..." << endl;

return 0;

}

**Acceso a los datos miembro usando punteros.**

* Para acceder a los datos y funciones miembro de objetos Gatos creados localmente, usamos el operador punto (.).
* Para acceder a objetos Gato en el heap, debemos desreferenciar el puntero y llmar el operador punto en el objeto apuntado por el puntero.

(\*pMiso).getEdad();

**Acceso a los datos miembro usando punteros**

* Los paréntesis se utilizan para asegurar que pMiso se desreferencie antes de que GetEdad sea accedido. *Se hace con paréntesis porque las operaciones en paréntesis se realizan antes que lo que no esté dentro del paréntesis.*
* Como esto es un poco confuso, el C++ provee un operador para acceso indirecto: el operador apunta-a que se representa ->

**Datos miembro en el heap**

* Uno o más de los datos miembro de una clase puede ser un puntero a un objeto en el heap.
* La memoria puede ser asignada en el constructor de la clase o en una de sus funciones y puede ser borrada en su destructor.

**El puntero this**

* Cada función miembro de una clase tiene un parámetro oculto: el puntero this.
* this apunta al objeto individual.
* En cada llamada a getEdad() o a setEdad(), el puntero this del objeto es incluido en un parámetro escondido.
* El trabajo del puntero this es apuntar al objeto individual cuya función ha sido invocada.
* Generalmente no necesitamos this; solo llamamos las funciones y hacemos set a las variables miembro.
* A veces necesitamos acceder al objeto en si mismo y es entonces cuando el puntero this se vuelve útil.
* Normalmente no necesitamos usar el puntero this para acceder a las variables miembro de un objeto desde las funciones del objeto.

Punteros colgantes

Llamados en ingles stray, dangling o wild.

Recordar que una vez que se ha borrado un puntero debe ser resignado a nullptr para evitar que el programa se falle.

Punteros constantes

Utilizando la palabra clave const podemos crear punteros constantes de tres formas distintas:

*const int \*pUno;*

*int \* const pDos;*

*const int \* const pTres;*

En la segunda declaracion pDos es un puntero constante. El valor al cual apunta no puede ser cambiado usando el puntero:

*const int \*pUno;*

Por lo que NO es posible escribir:

\*pUno = 5; // Esta asignacion produciria un error.

En la segunda declaracion pDos es un puntero constante a un entero. El entero puede cambiarse pero el puntero pDos no puede ser reasignado:

*int \* const pDos;*

Por lo que NO es posible escribir:

pDos = &x; // Esta asignacion produciria un error.

En la tercera declaracion pTres es un puntero constante a un entero constante. El entero no puede cambiarse y el puntero pTres no puede ser reasignado:

*const int \* const pTres;*

Por lo que NO es posible escribir:

\*pTres = 5; // Esta asignacion produciria un error.

pTres = &x; // Esta asignacion produciria un error.

Punteros constantes y funciones miembro constante

Si declaramos un puntero a un objeto constante, las unicas funciones que podemos llamar con ese puntero son las funciones constantes.

Ejercicio

Añade un gato llamado Misifus al programa creaheap.coo

Modifique el programa AccedeHeap de forma tal que no utilice el operador apunta-a.