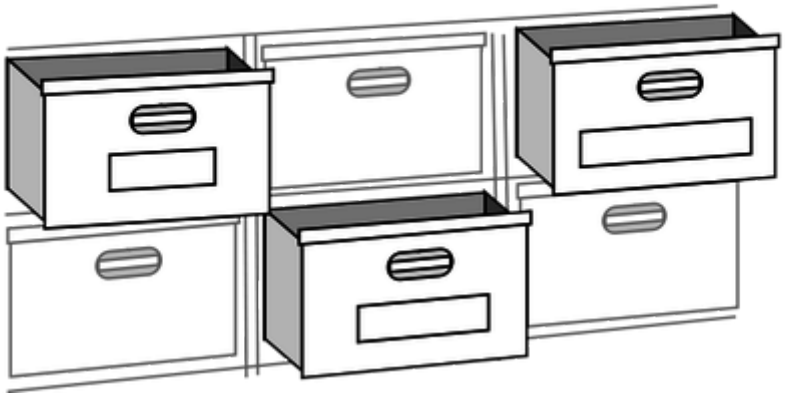
HDC

La verdad es que siento la verdadera necesidad de enseñarles a programar aunque exactamente no sea seguridad, pero sabiendo que es completamente inevitable aprender para ser un buen experto. Eso sí, no estoy haciendo completo el curso de C por lo que me gustaría que lo complementaran con otros cursos. Yo dejaré algunos links abajo :). Reforzaremos lo visto la última clase de C y veremos cosas nuevas.

Bueno, para comenzar lo que veremos es algo denominado "**variable**". Las variables son **pequeños segmentos de código reservados por el compilador a pedido de nosotros donde podemos almacenar valores**. Algo así como un **cajón** donde **guardamos** un **valor**. Estos cajones están en el armario que nosotros llamamos **memoria**.



Ahora supongamos que los cajones están divididos por **bytes** (recuerden que 1 byte

= 8 bits) y con una variable podríamos necesitar más de un byte porque nuestro valor es más grande que lo que entra en un cajón. En este caso llamaremos variable a ese **conjunto de cajones**. Repito que **el compilador se encarga de asignar cajones** en este caso y entonces nos despreocupamos. Sólo no se olviden la teoría.
Nuestro trabajo se limita a decirle **qué necesitamos**, por ejemplo: un espacio de memoria de X cantidad de bytes donde vamos a guardar un valor entero, real, con signo o de otra índole. **¿Cómo hacemos esto?** Con esta **estructura**:

```
(TipoVariable NombreVariable = ValorVariable;)
```

Entonces, primero sabremos **qué valor almacenar** y de ahí elegiremos nuestro **tipo de variable**. En la tablita están todos los tipos:

Tipo	Tamaño	Rango
unsigned char	1B	[0,255]
char	1B	[-127,127]
short int	2B	[-32768,32768]
unsigned int	4B	[-2’.147.483.648,2’.147.483.648]
unsigned long	4B	[0, 4.’294.967.295]
enum	2B	[2’.147.483.648, 2’.147.483.648]
float	4B	[3.4x10-38,3.4x10+38]
double	8B	[1.7x10-308,1.7x10+308]
long double	9B	[3.4x10-4932,3.4x10+4932]
bool	1B	true, false

Para cada uno tenemos su **rango de valores** correspondientes y su **peso en memoria**. Para los que quieren hacer buenas prácticas de programación (en este curso, me imagino que todos :P), tienen que usar la **menor cantidad de variables con el menor peso posible**. Supongamos que quiero almacenar dos valores enteros, un número con coma y dos valores lógicos.

Es decir que nos quedaría así:

```
main()
{
	int primerNumero = 21, segundoNumero = 34;
	float numeroConComa = 21.3;
	bool primerValorLogico = true;
	bool segundoValorLogico = false;
}
```

Fíjense que para declarar dos variables del mismo tipo lo puedo hacer separadas de una coma como lo hice con los valores enteros. Aunque para ser prolijos, es preferible hacerlo **una variable debajo de otra** como con los valores lógicos. Otro consejo que tienen que saber es que no es indispensable darle un valor a la variable cuando la declaramos (porque muchas veces no sabemos qué es lo que vamos a guardar allí dentro) pero es buena práctica hacerles un espacio un poco más abajo donde vamos a asignarles los correspondientes y no en el mismo lugar donde las creamos.

```
main()
{
	int primerNumero, segundoNumero;
	float numeroConComa;
	bool primerValorLogico;
	bool segundoValorLogico;

	primerNumero = 21;
	segundoNumero = 34;
	numeroConComa = 21.3
	primerValorLogico = true;
	segundoValorLogico = false;
}
```

Ahora, para seguir un poco más adelante, veremos lo que son las **funciones** y los **procedimientos**. Aunque hoy en día todo es llamado función, o por lo menos siempre lo veo por manuales y libros, existe una diferencia entre ambos.
Cuando hablamos de una **función** o **procedimiento**, estamos diciendo que saltamos a un **mini programa** fuera de la función donde estamos ejecutando el código -que es la función main-. Y los separamos: **la función devuelve un valor específico y el procedimiento no devuelve ningún valor** luego de terminar esa pequeña porción de código. Por ejemplo:

```
main()
{
	//declaracion de variables
	int numero1;
	int numero2;
	int resultado;

	//asignacion
	numero1 = 2;
	numero2 = 3;

	//programa
	resultado = suma(numero1, numero2);
	printf("El resultado es: %i", resultado);
}
```

La función suma no existe, es sólo un ejemplo

En la primera linea tenemos la función "***suma***" que se encarga de sumar dos números y devuelve el resultado para guardarlo en otra variable. En cambio, "***printf***" es un procedimiento que se encargará de mostrar un texto en pantalla pero no devuelve nada, es una simple ejecución de instrucciones fuera del código principal.

Pero ¿para qué usamos esto?

Esto sirve para 2 cosas principales:

1. **Para ser prolijo**. Es decir, para no tener una cantidad monstruosa de código en el main y que para buscar dónde había hecho uno la parte donde calculamos la distancia entre dos puntos, sino que directamente llenamos al main de llamadas a funciones y procedimientos externos y luego podemos directamente ir a esas funciones separadas. Piensen que hay programas con miles de decenas de líneas. **Para resumir: es una buena práctica de programación para poder encontrar errores fácilmente**.

2. P**ara no repetir muchas veces las mismas series de instrucciones.** Porque hay veces que tenemos que hacer lo mismo una y otra vez, no quiere decir que tenemos que copiar diez líneas de código iguales. Además supongamos que sabemos que ese pedazo de código tiene un error y queremos resolverlo. Mejor arreglarlo una sola vez dentro de la función que diez veces en cada lugar donde se copio ese fragmento.

En fin, es útil y lo verán mucho. Veamos la sintaxis para la creación de funciones y procedimientos:

```
tipoValorRetorno nombreFuncion (tipoVariable nombreVariable, tipoVariable2 nombreVariable2...)

{
        codigo
}
```

Vamos por partes, dijo Jack el destripador. En el tipo de valor de retorno es donde se define si este fragmento se comportará como función o como procedimiento. En caso de **no** tener un **valor de retorno** (esto quiere decir que genera un valor al terminar el fragmento) usamos la palabra **void**, haciendo lo que llamamos **procedimiento** y **si tiene algun valor** como int o float, o algún otro como hemos visto en la tabla de variables, será una **función**.

Veamos si hubiésemos creado un **procedimiento**:

```
void miProcedimiento (void)
{
      printf("Estoy dentro del procedimiento");
}

int main (void)
{
      miProcedimiento();
}
```

El main debe retornar siempre un valor entero

Primero tengo que **declarar el procedimiento** y luego hago la **llamada desde el main**. La llamada se hace **obligatoriamente con los paréntesis**, aunque estén vacíos como en este ejemplo porque no hay que pasar parámetros.
Igualmente este programa, **si lo intentamos de compilar, no funcionará**. ¿Por qué? Por la razón de que la función **printf no está definida**. ¿La tenemos que definir nosotros? Bueno, no. Se dieron cuenta que hay funciones que se usan para más de un programa repetidas veces. Entonces existen lo que denominamos **librerías** que son **porciones de código** donde se **definen** muchas **funciones** y uno puede **invocar** estas **librerías** para tener la posibilidad de usar todo lo que tienen dentro,. La librería de printf es **stdio.h** (el .h es la extension de la librería). Y perdón por seguir alargando el párrafo pero se llama asi por "**standard in/out**" y no por "studio" xD -lo escuché más veces de lo que creen-.

Pero para **llamar a una librería** lo hacemos arriba de todo, en las primeras líneas de nuestro programa. Con esta **sintaxis**:

```
#include <libreria>
```

Fácil de recordar. **Nuestro programa** quedaría de esta manera:

```
#include <stdio.h>

void miProcedimiento (void)
{
     printf("Estoy dentro del procedimiento");
}

int main (void)
{
     miProcedimiento();
}
```

Ahora podemos **compilar** (Execute → Compile o ctrl+F9) y darle run desde la consola para ver si funciona

```
C:\Users\w7\Desktop>prueba.exe
Estoy dentro del procedimiento
C:\Users\w7\Desktop>
```

Todo perfecto y sin errores. En caso de haber hecho algo mal habrá bugs o seguramente errores cuando compilamos.
Vamos a crear un procedimiento nuevo pero que necesite dos parámetros. **Simplemente sumará dos números y los mostrará en pantalla.** Nuestro **procedimiento** entonces sería de esta manera:

```
void sumarEnteros (int entero1, int entero2) 1
{
     int resultado; 2
     resultado = entero1 + entero2;
   3 printf("El resultado es %i", resultado);
}
```

Paso a paso:

1. Tenemos 2 parámetros ahora que necesitamos pasar (que serían los dos números a sumar) con sus respectivos tipos de variables.
2. Dentro del procedimiento, creamos una variable donde almacenaremos

el resultado. Esta variable no sigue existiendo luego del procedimiento.

3. Mostramos en pantalla el resultado que es un valor entero almacenado en la variable “resultado”. Para ésto, printf necesita que le pasemos un parámetros de esa característica. Más adelante profundizaremos.

El programa entero ahora sería así:

```
#include <stdio.h>

void sumarEnteros (int entero1, int entero2)
{
      int resultado;
      resultado = entero1 + entero2;
      printf("El resultado es %i", resultado);
}

int main (void)
{
     int numero1;
     int numero2;

     numero1 = 24;
     numero2 = 25;

     sumarEnteros (numero1, numero2);
}
```

Ya estamos bastante bien con respecto a programar si están entendiendo todo. Recuerden que **siempre se empieza a ejecutar la primera línea de la función main**. Entonces primero **creamos las variables** que serán nuestros números a sumar, luego **les damos un valor** y por último **invocamos a la función** que creamos anteriormente con los parámetros necesarios.

Fíjense que cuando llamamos a la función le damos dos variables con nombres distintos a los que usa la función misma. Esto porque el procedimiento en realidad, **crea nuevas variables** (aunque pueden tener exactamente el mismo nombre, como si se tratase de un programa a parte) donde almacenará el valor de las variables que le pasamos. En este caso serían los valores 24 y 25. Luego jugamos con las variables creadas en ese procedimiento pero no afectamos las variables originales del programa principal dentro de la función main... Uf! Que explicar programación puede ser denso ¿Verdad?

Bueno veamos **luego de compilarlo** si es que nos funcionó:

```
C:\Users\w7\Desktop>prueba.exe
El resultado es 49
C:\Users\w7\Desktop>_
```

Todo perfecto. Vamos a hacer un ejemplo más antes de terminar que será la creación de una función. **Esta función devolverá el resultado de la suma** (reciclaremos el procedimiento anterior). Veamos la diferencia:

```
#include <stdio.h>
int sumarEnteros (int entero1, int entero2)
{
     int resultado;
     resultado = entero1 + entero2;
     return resultado;
}

int main (void)
{
    int numero1;
    int numero2;
    int resultado;

    numero1 = 24;
    numero2 = 25;

    resultado = sumarEnteros (numero1, numero2);

    printf ("El resultado es %i", resultado);
}
```

1. Cambiamos el tipo de retorno de "***void***" a entero porque al ser una función queremos que nos devuelva el resultado.
2. En esta línea vemos el comando ***return***, el cual sirve para aclarar la variable de retorno de la función
3. Allí pasamos los parámetros a la función y como nos va a devolver un valor, debemos guardarlo en una variable como si fuera el resultado de una cuenta matemática. No era necesario que la variable se llame igual que la de retorno porque lo que retorna es el valor almacenado dentro de la variable y no el nombre de la variable.
4. Mostramos en pantalla el resultado final.

En fin, vamos a dejar esta clase aquí. Quiero que intenten de crear cosas con esto que aprendimos. Sé que es básico y aburrido, pero piensen que pueden hacer como una calculadora o algo que vemos cotidianamente para practicar programación:).

Pueden seguirme en Twitter: @RoaddHDC

Cualquier cosa pueden mandarme mail a: r0add@hotmail.com

Para donaciones, pueden hacerlo en bitcoin en la dirección siguiente:
1HqpPJbbWJ9H2hAZTmpXnVuoLKkP7RFSvw

Roadd.
