Contenido

3. CONCEPTOS BASICOS I

Es sabido que existen muchas diferencias, y la mayoría de las veces grandes diferencias, entre el lenguaje C y C++. Aunque muchas de estas diferencias recaen en la funcionalidad de los códigos, mas no en su sintaxis. Por ejemplo, si tenemos un programa escrito en C, podemos decirle a un compilador de códigos de C++ que lo haga funcionar, sin embargo, es posible que lo inverso que no se pueda hacer tan fácilmente.

En principio su sintaxis es la misma, la mayor diferencia son algunas funcionalidades que pueden ser mejores en C++. En esta parte del material haremos uso de ambas sintaxis, haciendo énfasis en las diferencias, lo cual veremos que son mínimas para nuestros propósitos.

Para comenzar nuestro aprendizaje sobre las instrucciones que debemos aprender los conceptos básico sobre el lenguaje C/C++, usaremos el **código 2.1** visto en las notas **Primeros pasos**. Nuestro objetivo será analizar este pequeño código para introducir los primeros conceptos. Al final de estas notas, se encuentra un breve resumen sobre las instrucciones utilizadas.

3.1 Hola mundo. Como funciona.

Los **headers**, como hemos mencionado anteriormente, son un conjunto de utilerías que pedimos sean *agregadas* a nuestros código para utilizarlas en el futuro. En particular, al agregar las líneas `#include <stdio.h>` o `#include <iostream>` le pedimos al compilador que agregue la posibilidad usar dos importantes funciones: mostrar información en la pantalla (como el texto ***Hola mundo***) y poder solicitar información al usuario a través del teclado (esta función se discutirá en futuras secciones). Aunque esta primera línea es muy sencilla, será indispensable de aquí en adelante, además de que nos permitirá usar más de una de sus herramientas útiles.

La siguiente instrucción es `int main()`. Con ella indicamos al compilador que vamos a comenzar a darle instrucciones para realizar algún proceso, ya sea, imprimir a la pantalla, pedir información o

calcular al dato. El compilador entenderá que lo que este después de esta instrucción será todo lo que necesitemos hacer. Esta instrucción al igual que la anterior, no podemos cambiarla, **siempre debemos escribirla de la misma forma**.

Aquí es donde daremos uno de los conceptos más importantes, tanto en el ámbito de programación, como lo será en el lenguaje C/C++. Después de escribir la instrucción ***`int main()`*** hemos de escribir los símbolos "**{**" y "**}**"en este orden, que llamaremos **llaves**. Cuando se escribió el **código 2.1**, se realizó de la siguiente manera:

Primero se escribió el **header** correspondiente y la instrucción ***`int main()`*** seguida de los símbolos llaves (ver **código 3.1**). Al primer símbolo "**{**" le decimos de llave de apertura y de cerradura al segundo símbolo "**}**".

```c
#include <stdio.h>

int main()
{

}
```

```cpp
#include <iostream>

int main()
{

}
```

Código 3.1. Códigos iniciales en ambos lenguajes. En ocasiones son nombrados ***esqueletos.***

Cuando se tiene el **esqueleto** del código anterior, en el que se ha dejado intencionalmente una línea vacía entre los símbolos **llaves**. Es entre estos dos símbolos donde debemos escribir las instrucciones siguientes. **La instrucción que agregaremos es *`printf()`* que puede ser usada en dos formas distintas. La primera y más sencilla es escribir entre los paréntesis el texto que queremos mostrar agregando al inicio y al final de este texto comillas dobles.**

Para realizar la misma acción **en lenguaje C++, debemos utilizar *`std::cout <<`* seguida del texto que queremos imprimir agregando nuevamente las comillas dobles** como en la instrucción ***`printf()`***. Al terminar de escribir nuestro texto, sin importar cual instrucción se uso, debemos terminar con el símbolo "**;**" (**punto y coma**). Este símbolo, representa que hemos terminado una instrucción. En el futuro, cuando usemos mas instrucción, veremos que el símbolo **punto y coma** tiene la finalidad de ***separar instrucciones***.

```c
#include <stdio.h>

int main()
{
    printf("Hola mundo");
}
```

```cpp
#include <iostream>

int main()
{
    std::cout << "Hola mundo";
}
```

Código 3.2. Se han agregado entre los símbolos **llaves** y las instrucciones necesarias para mostrar el mensaje **Hola mundo** en la pantalla.

Finalmente, cuando ya no necesitemos realizar ninguna otra acción debemos incluir la instrucción **`return 0`** (agregando el símbolo "**;**" al final) para indicarle al compilador que ya no requerimos realizar más instrucciones. Esta instrucción es como un **stop** para nuestro código y debemos aprenderla y no olvidarla.

```c
#include <stdio.h>

int main()
```

```cpp
#include <iostream>

int main()
```

```
{
    printf("Hola mundo");
    return 0;
}
```

```
{
    std::cout << "Hola mundo";
    return 0;
}
```

Código 3.3. Código terminado en ambos lenguajes.

Aquí definiremos un nuevo concepto, que es conocido en el ambiente de programación: **Al conjunto de instrucciones que se encuentran entre de los símbolos llave le llamaremos bloque de instrucciones.** En particular, para la instrucción ***int main()*** decimos**, el bloque de instrucciones que pertenece a la instrucción *int main()*.**

Ejercicio: *prueba cambiando el mensaje que aparece en la pantalla*

3.2 Primeros ejemplos

Como en cualquier material que aprendemos, es importante comenzar con ejercicios que pueden ayudarnos a entender más el material mismo. El ejercicio proveerá de preguntas y errores, que al resolverlos ayudarán a comprender aún más las instrucciones que ya conocemos.

Te sugerimos que para cada código nuevo, crear un archivo nuevo y guardarlo como lo hemos hecho en la sección 2.1.

3.2.1 Imprimir un rombo

Como primer ejemplo, utilizaremos las funciones ***printf()*** o ***cout <<*** para crear alguna "imagen" usando solo símbolos de texto. Nuestro objetivo será crear un *rombo*.

```
#include <stdio.h>

int main()
{
    printf("  *  ");
    printf(" *** ");
    printf("*****");
    printf(" *** ");
    printf("  *  ");
    return 0;
}
```

```
#include <iostream>

int main()
{
    std::cout << "  *  ";
    std::cout << " *** ";
    std::cout << "*****";
    std::cout << " *** ";
    std::cout << "  *  ";
    return 0;
}
```

Código 3.4. Código para imprimir un rombo sencillo.

Lo que hacemos en el **código 3.4** es "**imprimir**" (otra forma para decir mostrar) primero 5 caracteres, donde los 2 primeros son espacios, después 1 asterisco, y después 2 espacios. Después imprimimos 1 espacio, 3 asteriscos y luego 1 espacio, y así sucesivamente. Pero hay pequeño problema con nuestro código; veamos que sucede cuando lo compilamos y lo ejecutamos. La imagen 3.1 nos muestra el resultado cuando "**corremos**" (otra forma de decir ejecutar) nuestro código. Si no recuerdas como hacerlo, revisa la **sección 2.2.**

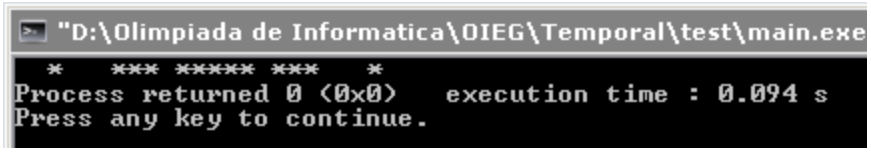

Imagen 3.1. *Resultado de ejecutar el **código 3.4**. Observamos que no obtenemos el resultado deseado.*

Lo que ha pasado es que tenemos un ***error de ejecución***, es decir, que el código se ejecuta correctamente pero no obtenemos el resultado que queremos. El problema es difícil por ahora de encontrar dado que estamos iniciando. Primero veamos cómo es que las instrucciones se realizan: dado que hemos escrito 6 instrucciones (5 `printf()` y 1 `return 0`) **¿en qué orden se ejecutan estas instrucciones?** O una mejor pregunta sería: **¿en qué orden se ejecutan las instrucciones de cualquier código dado?** Ambas preguntas son similares, pero es la segunda la que está hecha de manera más general., entonces démosle respuesta.

La respuesta más intuitiva es este caso, es la más acertada. Las instrucciones en un código se ejecutan en el orden en que son dadas, esto es, que se ejecutan de manera secuencial de arriba hacia abajo. Para el **código 3.4**, primero se ejecutara la instrucción `printf("  *  ")`, es decir la que tiene 1 asterisco, después la instrucción que tiene 3 asteriscos, enseguida la que tiene 5 asteriscos, a continuación la instrucción con 3 asteriscos, luego la instrucción con 1 asterisco y finalmente la instrucción `return 0`.

Veamos lo que hemos hecho en el **código 3.4**: nuestra primera instrucción que se ejecuta es `printf("  *  ")`, como ya vimos antes, permite imprimir 2 espacios, 1 asteriscos y 2 espacios, pero que pasa cuando se ejecuta la segunda instrucción `printf(" *** ")`, le hemos dicho que imprima 1 espacio, 3 asteriscos y 1 espacio. Pero **¿en qué momento le dijimos que imprimiera debajo de donde se imprimió el texto de la instrucción anterior?** No hemos indicado esto, simplemente le dijimos que imprimiera. El compilador ha entendido lo siguiente: imprimir el texto `"  *  "`, después quieres que imprima el texto `" *** "`, luego que imprima el texto `"*****"`, etc. **El compilador no ha recibido alguna instrucción que diga**: imprime un texto y pasa a la siguiente línea, imprime otro texto y pasa a la siguiente línea, etc.

Por lo anterior es que no vemos en la **pantalla negra**, la imagen del rombo. Para corregir nuestro **error de ejecución**, debemos indicarle que después de imprimir el texto se pase a la siguiente línea: **agreguemos al final de cada texto, antes de la segunda comilla doble, los dos caracteres** `"\n"` (los caracteres van juntos, no lo olvides). Este símbolo significan "**new line**", entonces lo que le pedimos al compilador es: imprime el texto `"  *  "` y después pasa a la siguiente línea. Entonces el **código 3.4** lo debemos modificar para obtener el **código 3.5**.

```c
#include <stdio.h>

int main()
{
    printf("  *  \n");
    printf(" *** \n");
    printf("*****\n");
    printf(" *** \n");
    printf("  *  \n");
    return 0;
}
```

```cpp
#include <iostream>

int main()
{
    std::cout << "  *  \n";
    std::cout << " *** \n";
    std::cout << "*****\n";
    std::cout << " *** \n";
    std::cout << "  *  \n";
    return 0;
}
```

Código 3.5. *Código para imprimir un rombo sencillo.*

Cuando nos referimos a "la siguiente línea", lo que tratamos de decir es que en lugar de continuar imprimiendo en el mismo renglón (fila o línea), continúe haciéndolo en la línea que se encuentra justo debajo. Algo importante que debemos saber es que cuando pedimos que se pase a la siguiente línea (o más comúnmente salto de línea) entonces cuando se imprima información esta se imprimirá desde el inicio (desde la izquierda).

Es importante mencionar que el símbolo `"\"` no es el mismo, que la diagonal `"/"` conocida para dividir. **El símbolo `"\"` es conocido como diagonal inversa y es un símbolo especial**.

Ejercicio: *Intenta poner la diagonal normal en lugar de la diagonal inversa y observa el resultado. Si cambiamos la primera y penúltima instrucción por* `printf("  *\n")` *(imprimir 2 espacios, luego 1 asteriscos y pasar a la siguiente línea).* ***¿Hay algún cambio? ¿Por qué?***

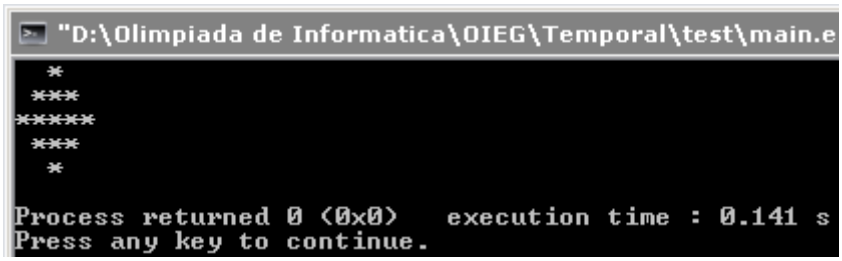

Imagen 3.2. *Resultado obtenido al ejecutar el código 3.4.*

Ejercicio*: trata de realizar el código para hacer un rombo más grande*

3.2.1 Primera letra de tu nombre

Puedes hacer un código que imprima la imagen de la primera letra de tu nombre usando símbolos "**#**" en lugar de asteriscos. Dado que el nombre OIEG empieza con la letra O, haremos el código que genera una O gigante usando # y espacios.

```c
#include <stdio.h>

int main()
{
    printf("######\n");
    printf("##  ##\n");
    printf("##  ##\n");
    printf("##  ##\n");
    printf("######\n");
    return 0;
}
```

```cpp
#include <iostream>

int main()
{
    std::cout << "######\n";
    std::cout << "##  ##\n";
    std::cout << "##  ##\n";
    std::cout << "##  ##\n";
    std::cout << "######\n";
    return 0;
}
```

Código 3.6. *Código para mostrar una O gigante formada por* ***#*** *y* ***espacios****.*

Ejercicio*: prueba imprimiendo otras letras (una a la vez)*

3.3 Funciones I

Para iniciar en este concepto, imaginemos que queremos imprimir más de una letra O como la que se hizo en el **código 3.6**, entonces tendríamos que hacer algo parecido al **código 3.7**. Lo que podemos observar es que copiamos el texto correspondiente a las instrucciones para imprimir una O, una y otra vez, agregando en medio de estas **la instrucción *`printf("\n")`* para imprimir una línea vacía** entre cada O. Lo que obtenemos es tres O's gigantes (una debajo de otra) y una línea entre cada una.

También es importante hacer notar que podemos dejar líneas vacías cuando escribimos instrucciones, por ejemplo, vemos que el ***`return 0`*** se encuentra dos líneas después de la ultima instrucción ***`printf()`***. El compilador entenderá que ese espacio entre las instrucciones no afecta la ejecución; más aún, el compilador reconoce que sin importar cuantos espacios o líneas vacías allá entre 2 instrucciones, después de una instrucción se encuentra la siguiente instrucción.

Entonces, **¿qué sucedería si quisiéramos imprimir muchas O?** o si quisiéramos imprimir otras letras gigantes. Pues hasta ahora, nuestra única solución seria, copiar tantas veces como lo necesitemos. Sería perfecto poder decirle al compilador "**imprime una O**" y que el compilador entendiera que debe imprimir una O. **Pues es precisamente de este tipo de ejemplos, que podemos recurrir a la creación de funciones.**

```c
#include <stdio.h>

int main()
{
    printf("######\n");
    printf("##  ##\n");
    printf("##  ##\n");
    printf("##  ##\n");
    printf("######\n");
    printf("\n");
    printf("######\n");
    printf("##  ##\n");
    printf("##  ##\n");
    printf("##  ##\n");
    printf("######\n");
    printf("\n");
    printf("######\n");
    printf("##  ##\n");
    printf("##  ##\n");
    printf("##  ##\n");
    printf("######\n");


    return 0;
}
```

```cpp
#include <iostream>

int main()
{
    std::cout << "######\n";
    std::cout << "##  ##\n";
    std::cout << "##  ##\n";
    std::cout << "##  ##\n";
    std::cout << "######\n";
    std::cout << "\n";
    std::cout << "######\n";
    std::cout << "##  ##\n";
    std::cout << "##  ##\n";
    std::cout << "##  ##\n";
    std::cout << "######\n";
    std::cout << "\n";
    std::cout << "######\n";
    std::cout << "##  ##\n";
    std::cout << "##  ##\n";
    std::cout << "##  ##\n";
    std::cout << "######\n";


    return 0;
}
```

Código 3.7. *Código para mostrar una O gigante formada por* ***#*** *y* ***espacios****.*

Las funciones las podemos ver como instrucciones, que realizan un grupo definido de instrucciones. Aunque puede sonar un poco confusa la definición, lo que queremos decir es que las funciones agrupan un conjunto de instrucciones. Por otro lado, dado que no existe en el lenguaje natural de C/C++ una instrucción para imprimir O's gigantes, entonces debemos **crear una instrucción** que haga eso, a esta instrucción le llamaremos **la función para imprimir O's gigantes**.

Podemos describir en pocas palabras que: ***el objetivo de las funciones es particularmente, ejecutar un bloque de instrucciones la cantidad de veces necesarias, sin el trabajo de escribir repetidamente el mismo bloque una y* otra vez.**

Antes de comenzar debemos saber que **existen 2 tipos de funciones, *funciones sin parámetros* y *funciones con parámetros***. **Una función con parámetros** es aquella que **requiere** de uno o varios valores para funcionar; por ejemplo, la primera función con parámetros que hemos visto, aunque no le hemos llamado propiamente función, en la instrucción ***printf()*** que requiere como parámetro el texto que deseamos aparezca en pantalla, tal como lo hemos hecho con la frase **hola mundo** o con un **salto de línea solamente**, como en el **código 3.7** en la instrucción ***printf("\n")***.

El contenido alrededor de las funciones es amplio, por lo que es importante abordarlo de manera sencilla. Comencemos entonces, utilizando **funciones sin parámetros** del modo sencillo**.**

3.3.1 Declaración de funciones. Creación de una función sin parámetros

Al contrario de las funciones con parámetros, las funciones sin parámetros no necesitan de información extra para funcionar. Las funciones sin parámetros son un buen inicio, sin embargo, es necesario incluso iniciar de una manera más simple.

Para indicarle al compilador que queremos una nueva instrucción (función), debemos hacerlo a través de nuestro código. Existe un lugar determinado para hacerlo: el cual es entre la sección **headers** y la sección **código principal** (revisar la **sección 2.1** y **3.1**).

En nuestro código, debemos agregar las siguientes instrucciones:

```
void nombre_nueva_instruccion();
```

La palabra ***void debe ser escrita con minúsculas***. En lugar de **nombre_nueva_instruccion**, debemos sustituir por el nombre que queremos asignarle a la nueva instrucción. Los símbolos que se encuentran después del nombre, son el **paréntesis de apertura** "**(**" y el **paréntesis de cerradura** "**)**" y al final como siempre el punto y coma. Para nuestro ejemplo de las O's, un buen nombre seria **imprime O grande**, sin embargo hay un pequeño problema. En el lenguaje C/C++ hay un **estándar** respecto a cómo debemos nombrar las nuevas instrucciones. Las reglas son bastante simples:

- El nombre de la instrucción **DEBE** comenzar con una letra (mayúscula o minúscula) o guión bajo.
- Solo puede contener los símbolos tipo letra (cualquier letra en el **alfabeto inglés** mayúscula o minúscula), números y/o guiones bajos. **Los espacios no están permitidos**.

El nombre que hemos elegido, ha infringido en la segunda regla, pues requerimos espacios para nuestro nombre. Pero podemos arreglarlo simplemente eliminándolos, entonces el nombre de nuestra instrucción será: **imprimeOgrande**, y así no habremos de infringir regla alguna. Algunos autores, prefieren utilizar el "**guion bajo**" como un **sustituto al símbolo de espacio**; si siguiéramos la sugerencia de estos autores el nombre de nuestra instrucción seria: **imprime_O_grande**. Al final será decisión de cada usuario, cual idea seguir de aquí en adelante. Particularmente, sugerimos la segunda forma.

Entonces el código para crear nuestra función, seria:

```
#include <stdio.h>          Headers          #include <iostream>
```

```
void imprime_O_grande();

int main()
{
    return 0;
}
```

Declaración de funciones

Código principal

```
void imprime_O_grande();

int main()
{
    return 0;
}
```

Código 3.8. *Código para indicar que queremos una nueva instrucción llamada* ***imprime_O_grande.***

De ahora en adelante, en lugar de llamar a **imprime_O_grande** la nueva instrucción, **le llamaremos la "función imprime_O_grande"**. En la **sección 2.1** hemos mencionado que un código en C/C++ contiene varias secciones, una de ellas es la sección de **declaración de funciones**; en la que se especifica el nombre de las funciones (o nuevas instrucciones) que **podríamos** utilizar en nuestro código y que son hechas por nosotros. Pues bien, nuestra función **imprime_O_grande** pertenece a esta sección, pues la utilizaremos en el **código principal** para imprimir O's grandes.

Ahora solo queda un paso muy importante: le hemos indicado al compilador que queremos una nueva función llamada **imprime_O_grande**, pero **también hace falta indicarle, que será lo que la nueva función debe de hacer cuando decidamos utilizarla**. De lo anterior, que la instrucción ***`void imprime_O_grande();`*** se encuentra en la sección **declaración de funciones**, ya que como su nombre lo indica, aquí solo hacemos mención de lo que podemos utilizar.

3.3.1 Definición de funciones.

En la **sección 2.1** se mencionó que la sección **definición de funciones** es **donde detallamos explícitamente que queremos que realicen las funciones declaradas en la sección declaración de funciones.**

Para **definir** la función **imprime_O_grande**, debemos hacerlo después del **símbolo llave de cerradura del código principal**. Una vez ahí, escribimos exactamente lo mismo que en la sección declaración de funciones, solo que esta vez no escribimos el símbolo punto y coma. En lugar de este ultimo símbolo escribimos ambos símbolos llave, el de apertura y después el de cerradura. **Lo que haremos será definir el bloque de instrucciones de la función** ***`void imprime_O_grande()`*** (ver final del **apartado 3.1) .** El **código 3.9** muestra cómo debemos realizar la definición de la función.

Si has escrito el **código 3.9** y utilizamos la función **F9** de **CodeBlocks**, observaras que no se imprime ninguna O grande formada por símbolos #'s. Esto pasa porque no hemos indicado en el **código principal** que queremos utilizar la función **imprime_O_gigante**. Lo único que debemos hacer es escribir en el código principal la instrucción ***`imprime_O_grande();`*** cada vez que queramos imprimir una O.

```
#include <stdio.h>

void imprime_O_grande();

int main()
{
    return 0;
}
```

```
#include <iostream>

void imprime_O_grande();

int main()
{
    return 0;
}
```

```c
void imprime_O_grande()
{
    printf("######\n");
    printf("##  ##\n");
    printf("##  ##\n");
    printf("##  ##\n");
    printf("######\n");
    printf("\n");
}
```

```cpp
void imprime_O_grande()
{
    std::cout << "######\n";
    std::cout << "##  ##\n";
    std::cout << "##  ##\n";
    std::cout << "##  ##\n";
    std::cout << "######\n";
    std::cout << "\n";
}
```

Código 3.9. *Código para **DEFINIR** la función **imprime_O_grande.** Observar que hemos eliminado el punto y coma para agregar el bloque de instrucciones de la función.*

3.3.1 Uso de las funciones definidas.

Cuando se han definido las funciones necesarias, solo nos resta definir donde y cuando las usaremos. Tomemos por ejemplo el **código 3.10** que al igual que el ejemplo anterior nos permite imprimir letras grandes.

Las funciones que se declaren (y definan) pueden ser usadas tantas veces como queramos o más comúnmente, cuando las necesitemos. Existen situaciones donde necesitamos utilizar la misma función más de una vez; aunque para nuestros primeros ejemplos será difícil mostrarlo. En futuras secciones veremos lo útiles que pueden ser cuando tratamos de resolver un problema.

```c
#include <stdio.h>

void imprime_A ();
void imprime_E_grande();
void imprime_I_grande();

int main()
{
    printf("Letras gigantes\n");
    imprime_E_grande();
    imprime_E_grande();
    imprime_A();
    imprime_I_grande();
    imprime_I_grande();

    return 0;
}

void imprime_E_grande()
{
    printf("######\n");
    printf("##\n");
    printf("####\n");
    printf("##\n");
    printf("######\n");
    printf("\n");
}

void imprime_I_grande()
{
    printf("######\n");
    printf("  ##\n");
    printf("  ##\n");
    printf("  ##\n");
    printf("######\n");
    printf("\n");
}
```

```cpp
#include <iostream>

void imprime_A();
void imprime_E_grande();
void imprime_I_grande();

int main()
{
    std::cout << "Letras gigantes\n";
    imprime_E_grande();
    imprime_E_grande();
    imprime_A();
    imprime_I_grande();
    imprime_I_grande();

    return 0;
}

void imprime_E_grande()
{
    std::cout << "######\n";
    std::cout << "##\n";
    std::cout << "####\n";
    std::cout << "##\n";
    std::cout << "######\n";
    std::cout << "\n";
}

void imprime_I_grande()
{
    std::cout << "######\n";
    std::cout << "  ##\n";
    std::cout << "  ##\n";
    std::cout << "  ##\n";
    std::cout << "######\n";
    std::cout << "\n";
}
```

```
void imprime_A()
{
    printf(" #### \n");
    printf("##  ##\n");
    printf("######\n");
    printf("##  ##\n");
    printf("##  ##\n");
    printf("\n");
}
```

```
void imprime_A()
{
    std::cout << " #### \n";
    std::cout << "##  ##\n";
    std::cout << "######\n";
    std::cout << "##  ##\n";
    std::cout << "##  ##\n";
    std::cout << "\n";
}
```

Código 3.10. *Código en el que se declaran 3 funciones llamadas **imprime_A()**, **imprime_E_grande()** e **imprime_I_grande()**. Notar que **las definiciones** están en **diferente orden**; el cual no afecta al funcionamiento final.*

En estos momentos podemos ver una de las principales utilidades más importantes que las funciones nos dan: nos permiten organizar lo que deseamos que se realice, es decir, podemos elegir cuando serán utilizadas, incluso podemos simplemente no utilizarlas si es que lo deseamos.

En el **código 3.10**, se han declarado 3 funciones, el orden en que las declaremos no importa así como tampoco el orden en que las definamos. Lo que es importante observar, es que las declaraciones se escriben antes del código principal y se definen después. Siempre se declaran una después de otra y se definen de la misma manera.

Ejercicio:	*prueba intercambiar el orden en que usamos las funciones (en el código principal) y observa el resultado. Utiliza más veces las funciones.*

Ejercicio:	*Analiza durante un tiempo sobre lo largo que sería escribir un código, sin usar funciones, para generar el mismo resultado y como este empeoraría si quisiéramos más de las mismas letras o incluso cambiar el orden de las letras.*

3.3.1 Flujo del código con funciones

El flujo de un código se refiere simplemente al orden en que las instrucciones se ejecutan. Cuando teníamos códigos sin funciones, sabíamos que las instrucciones se ejecutaban en el orden en que se escribían.

En el caso de funciones, no hay mucha diferencia, lo único que debemos hacer es **imaginarnos** que sustituimos cada función por las instrucciones que hay dentro de cada uno. Por ejemplo, en el código 3.10, cuando nuestro programa es ejecutado:

- El programa reconoce que en el **código principal**, la primera instrucción es imprimir el texto “Letras gigantes” y después un salto de línea.
- Después, el programa determina que tiene que ejecutar un bloque de instrucciones llamado ***`imprime_E_grande()`***. Para determinar que significa esta instrucción “**debe ir”** a la **definición de esta función** y ejecutar cada instrucción dentro de ella.
- Una vez que la ultima instrucción de la función ***`imprime_E_grande()`*** se ha ejecutado, entonces “**regresa**” nuevamente al **código principal** para ejecutar la

siguiente instrucción; y en este caso en particular, nuevamente se requiere hacer la instrucción ***imprime_E_grande()***, entonces se repite el proceso como antes.

- Cuando ha terminado de ejecutar por segunda vez la instrucción ***imprime_E_grande()***, el programa debe continuar ejecutando las instrucciones. Dado que las siguientes instrucciones son funciones, entonces realiza el mismo proceso de antes.
- La ultima instrucción en ejecutarse es utilizar la función ***imprime_I_grande()***. Cuando se han ejecutado por completo las instrucciones de la función, se ejecuta finalmente la ultima la instrucción ***return 0*** y termina nuestro programa.

Lo que debemos aprender es que cuando se **ejecutan las funciones**, una forma de verlo es como sustituir cada función por las instrucciones que hay en ella.

3.4 Resumen

- La estructura de un código de C/C++ tiene 4 principales componentes: **header**, **declaración de funciones**, **código principal** y **definición de funciones**.
- El **código principal** contiene las instrucciones que nuestro código ejecutara.
- Las instrucciones utilizadas en esta sección son ***printf()*** (***std::cout <<*** para C++) y ***return 0***. La primera nos permite imprimir un texto, agregando como parámetro el texto que queremos mostrar. La segunda es usada para terminar la ejecución de nuestro código.
- Al conjunto de instrucciones que se encuentran dentro de los símbolos **llaves** "**{**" "**}**", le llamamos **bloque de instrucciones**.
- Las funciones son instrucciones que contienen un grupo definido de instrucciones. Además pueden ser utilizadas siempre que lo necesitemos.
- Para crear un función debemos primeramente elegir un nombre que cumpla con las reglas: comenzar con letra o guión bajo, después de esto solo se puede usar letras, números y guion bajo. Una función debe declararse y definirse. Para declarar una función debe hacerse entre el **header** y el **código principal**. Para definirla lo hacemos después del código principal.
- Si requerimos mas funciones, estas deben se deben declaran una después de otra, siempre y cuando sea antes del código principal. La definición sigue el mismo principio, debe hacer después del código principal, definimos una función y después la siguiente.
- Para utilizar las funciones que hemos declarado y definido, solo necesitamos escribir como instrucción, el nombre de la función seguida de los paréntesis y el punto y coma, por ejemplo: ***imprime_A();***

3.5 Posibles errores

- Asegúrate de escribir correctamente los símbolos. Escribe correctamente las funciones ***printf()*** (o ***std::cout <<***). Recuerda son con minúsculas.
- Una importante sugerencia es que recuerdes como escribimos el código "Hola Mundo". Cuando escribimos los símbolos llaves, escribimos los dos símbolos, para asegurarnos que no se nos olviden. Cuando escribimos códigos, las llaves son los símbolos que más se olviden. Hacer lo anterior es una buena práctica para no olvidarlos.
- Si tienes una función declarada, asegúrate que al momento de definirla has escrito el nombre correctamente. Si declaramos una función llamada ***muestra_letras()***, entonces cuando la declaremos debe tener el mismo nombre.

- Recuerdo que cuando **declaramos funciones** ponemos el símbolo punto y coma. Cuando definimos funciones, ya no agregamos el símbolo punto y coma. Revisa los códigos **3.9** y **3.10**.
- Cuando utilices las funciones, debes escribir el nombre correctamente, para un compilador de C/C++ la función llamada `muestra_letras()` es diferente a la función llamada `muestra_Letras()` (la **l** es diferente). El compilador de C/C++ es sensitivo a mayúsculas y minúsculas, es decir, que las mayúsculas y las minúsculas son diferentes.