



industriales
etsii

Escuela Técnica
Superior
de Ingeniería
Industrial

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería
Industrial

Desarrollo de un Bot en la plataforma Azure para ayudar en el aprendizaje del lenguaje de programación C

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y
AUTOMÁTICA

Autor: Isidro Gómez Plasencia

Director: Diego Alonso Cáceres

Cartagena, 04/09/2019



Universidad
Politécnica
de Cartagena



Agradecimientos

Al igual que cualquier otra obra, un **Bot** como *aplicación informática* no está creado por los conocimientos de una sola persona, sino que se nutre de muchas otras partes como - otras personas, información de sitios web, libros, y demás -. Sin más demora agradecer a todos esas partes que hicieron posible este proyecto. En el apartado Bibliografía se recoge todo el material consultado.

Realizador

Isidro Gómez Plasencia

Profesor director del proyecto

Diego Alonso Cáceres

Contenido

Agradecimientos	1
Realizador	1
Profesor director del proyecto	1
1. Introducción	5
1.1. Propuesta del proyecto	5
1.2. Fases del proyecto	6
2. Estado de la técnica	7
2.1. Bot. Marco, Plataformas, Experiencia conversacional y Recorrido	7
2.1.1. Marco. ¿Qué es un Bot?	7
2.1.2. Plataformas. Administración y funcionamiento	8
2.1.3. Experiencia conversacional	11
2.1.4. Recorrido	12
2.2. Diseño del Bot en Azure	14
2.2.1. Funcionamiento de una Aplicación Bot	14
2.2.2. BOTAPRENDIZAJEC. Funcionamiento general	24
2.2.3. Herramientas de construcción	45
3. Desarrollo e implementación del Bot	47
BOTAPRENDIZAJEC. Funcionamiento detallado	47
3.1. Puesta en marcha de la Aplicación Bot	47
3.1.1. Archivo Program.cs	47
3.1.2. Archivo Startup.cs	48
3.2. Middleware	53
3.3. Punto de entrada de mensajería	54
3.3.1. Encauzamiento de los mensajes de entrada	56
3.3.2. Main	64
3.4. Gestor de respuestas	69
3.4.1. Archivos de recursos y archivos de notación	71
3.4.2. Respuestas simples	74
3.5. Recorrido de un punto del APRENDIZAJE DE C	74
3.5.1. Comienzo	75
3.5.2. Campos y constructor del diálogo en cascada	76
3.5.3. Primer paso de la cascada	77
3.5.4. Respuestas elaboradas	78
3.5.5. Control de interrupciones	79

3.5.6.	Segundo paso de la cascada	82
3.5.7.	Tercer paso de la cascada	84
3.5.8.	<i>CancelDialog</i>	84
3.6.	LUIS App. Administración y desarrollo	86
3.6.1.	Aplicación <i>LUIS</i>	87
3.6.2.	Panel	88
3.6.3.	Crear	89
3.6.4.	Administrar	90
3.6.5.	Entrenar	90
3.6.6.	Prueba	91
3.6.7.	Publicar	91
3.7.	Conversión del Resumen de C a tarjetas adaptativas	91
3.7.1.	Conversión de texto a código procesable	91
3.7.2.	Encapsulación del texto procesable en <i>tarjetas adaptativas</i>	93
4.	Conclusiones y trabajos futuros	97
4.1.	Conclusiones	97
4.1.1.	Funcionamiento	97
4.1.2.	Sensaciones	99
4.1.3.	Mejoras	102
4.2.	Trabajos futuros	103
4.2.1.	Versionado continuo del software	103
4.2.2.	Corto alcance	103
4.2.3.	Medio alcance	103
4.2.4.	Largo alcance	104
4.2.5.	Paralelismo	104
Bibliografía		104
Libros y Sitios Web		104
Temario de C		104
Problemas de C		104
Aprendizaje de C#		104
Aprendizaje sobre el Bot		105
Anexo		105
Contenido del TEMARIO		105
Temario. Objetivos, características y organización del contenido		105
PARTE I. METODOLOGÍA DE LA PROGRAMACIÓN		106
PARTE II. FUNDAMENTOS DE PROGRAMACIÓN EN C		109
Final del Temario de C		189
Contenido de los PROBLEMAS		190
Problemas. Características y organización del contenido		190
PARTE II. FUNDAMENTOS DE PROGRAMACIÓN EN C		191
Final de los Problemas de C		254



1. Introducción

1.1. Propuesta del proyecto

Antes de empezar con la explicación del proyecto, es necesario dar una explicación detallada sobre la propuesta del proyecto y su situación en el mercado actual.

La propuesta proponía el “Desarrollo de un Bot en la plataforma Azure para ayudar en el aprendizaje del lenguaje de programación C”, conteniendo todo lo necesario para un aprendizaje completo como explicación teórica, ejercicios resueltos y problemas de diversa índole. El Bot tendría que explicar de forma clara y eficaz cada punto y realizar una interacción con el usuario lo más rica posible.

Se quería desarrollar un Bot que enseñase el lenguaje C a los alumnos de 1º de grado de la UPCT.

Puesta la propuesta sobre la mesa y habiendo hecho una planificación previa, se expuso de forma general el contenido del lenguaje C que se pretendía implementar en el Bot:

1. **Lenguaje Básico.** Que comprendería: Fundamentos de la programación, elementos básicos del lenguaje, operadores y expresiones, estructuras de selección y estructuras repetitivas.
2. **Lenguaje Medio.** Que incluiría: funciones, arrays y cadenas.
3. **Lenguaje Avanzado.** Incluyendo: estructuras, memoria dinámica, pilas, colas y árboles.

Los puntos mencionados anteriormente incluyen no solo el texto plano sino definiciones, notas, avisos, titulaciones, énfasis en la escritura, ejemplos de uso, ejercicios resueltos, variedad de imágenes, etc en definitiva, todo lo necesario para una enseñanza completa.

En cuanto a la interfaz que el Bot mostraría al usuario, podría incluir menú, ayuda, índice, información sobre el lenguaje C, información de interés y otro aspecto que resultasen relevantes.

Por supuesto toda esta muestra de información no repercutiría negativamente en la *experiencia de usuario*, es decir, se crearía en base a un funcionamiento sencillo y eficiente que hiciera que el usuario no se sintiese frustrado.

Importante: El lenguaje Avanzado que en un principio se propuso para su implementación al final se desecharía por no ser preciso en el marco de la enseñanza educativa (alumnado) y estar más orientado al ámbito empresarial, aunque en un futuro podría ser interesante su implementación.

1.2.Fases del proyecto

Fases del Proyecto

TIEMPO TRANSCURRIDO		ACTIVIDAD
PERÍODO	TOTAL	
01/07/2018 – 10/8/2018	6 semanas	Resumen de C. Parte TEMARIO (teoría y ejercicios).
01/09/2018 – 08/10/2018	5 semanas	
09/10/2018 – 03/12/2018	8 semanas	Aprendizaje del lenguaje de programación C#.
04/12/2018 – 21/01/2019	7 semanas	Aprendizaje sobre la construcción de Bots de Microsoft, es decir, de la plataforma AZURE BOT SERVICE.
22/01/2019 – 28/02/2019	5.5 semanas	Construcción del Bot para el APRENDIZAJE DE C (BOTAPRENDIZAJEC). <ul style="list-style-type: none"> • Diseño, construcción previa, depuración. y pruebas. • Implementación Parte TEMARIO (teoría y ejercicios).
28/02/2019 – 19/03/2019	2.5 semanas	Resumen de C. Parte PROBLEMAS.
19/03/2019 – 20/03/2019	2 días	Corrección e implementación. Parte TEMARIO (teoría y ejercicios).
21/03/2019	1 días	Corrección. Parte PROBLEMAS.
22/03/2019 – 02/04/2019	1 semana y 5 días	Construcción del Bot para el APRENDIZAJE DE C (BOTAPRENDIZAJEC). Implementación Parte PROBLEMAS.
03/04/2019 – 19/04/2019	2 semanas y 3 días	Arreglos estructurales y de sintaxis, limpieza de código, arreglos UI, reorganización de modelos LUIS, ... Mejoras en general.
22/04/2019 – 22/07/2019	13 semanas y 1 día	Recopilación de toda la información precisa para la creación de proyecto. Realización de la Memoria del proyecto.

Nota: periodos computables 8 h/d de lunes a viernes y *algunos fines de semana*.

1.3.Organización del documento

La memoria está organizada en tres documentos Estado de la técnica, Desarrollo e implementación del Bot y Conclusiones y trabajos futuros que describen todo la información resumida que se ha requerido para proyectar, desarrollar y crear el Bot final propuesto en la Propuesta del proyecto.

- **Estado de la técnica:** proporciona toda la información necesaria para el perfecto conocimiento sobre la tecnología Bot y hace especial hincapié en la plataforma de Azure para la creación de Bots. Además, resume de manera general la funcionalidad del Bot creado (**BOTAPRENDIZAJEC**) y de las herramientas utilizadas.

- **Desarrollo e implementación del Bot:** Describe paso a paso toda la funcionalidad interna del Bot creado (**BOTAPRENDIZAJE**) para comprensión perfecta. Toda la información describe el proceso de desarrollo desde el comienzo hasta el final.
- **Conclusiones y trabajos futuros:** describe las conclusiones junto a las posibles mejoras, cambios, versionados de software, etc que el Bot podría sufrir de cara a un futuro.

2. Estado de la técnica

2.1. Bot. Marco, Plataformas, Experiencia conversacional y Recorrido

Importante: Toda la información explicada en este punto, **Bot. Marco, Plataformas, Experiencia conversacional y Recorrido**, se ha extraído de la documentación de Azure Bot Service de Microsoft y se ha explicado de manera diferente (más sencilla). La referencia a dicha documentación se sitúa en la Agradecimientos concretamente en Aprendizaje sobre el Bot.

2.1.1. Marco. ¿Qué es un Bot?

Un Bot es una aplicación o plataforma informática que realiza tareas automatizadas y con la cual el usuario interactúa para conseguir alguna respuesta. Puede ser tan simple como una respuesta basada en una serie de patrones en la pregunta o puede ser un tejido sofisticado mezclando inteligencia artificial con un seguimiento de estado de la conversación y la integración de servicios empresariales. Una pregunta puede usar desde cadenas de texto simples hasta *tarjetas enriquecidas* que contienen texto, imágenes, videos y botones de acción. Además, la conversación puede incorporar interacciones de lenguaje natural para permitir a los usuarios interactuar de manera natural con el Bot.

Nota: Un bot también puede comunicarse con otros Bots y de hecho lo hacen en redes de Bots o Bot Nets.

Entre los tipos de Bots se encuentran los Bots informativos, los Bots conversacionales y los Bots transaccionales, por otro lado, también hay Bots maliciosos que aquí no nombraremos.

Bot conversacional

El tipo de Bot usado será el **Bot conversacional**, por ser el que mejor se adapta a la propuesta del proyecto.

Usos: Este tipo de Bot se usa por ejemplo en la automatización de pedidos, solicitud de reservas en restaurantes, atención al cliente y solicitud de asistencia médica. Un ejemplo de atención al cliente sería la atención a preguntas frecuentes repetitivas, las cuales un bot haría de manera automática librándonos de tener a un operario con conocimientos de negocio contestando siempre las mismas preguntas.

Un bot es más que una simple herramienta de conversación, por todo lo dicho anteriormente y sumado a poder: conectarse a una lógica empresarial, conectarse a servicios externos (como

Servicios Cognitivos, de búsqueda, de recursos (almacenamiento), de telemetría, ...), integrarse en aplicaciones existentes, tener carácter proactivo, enriquecer la experiencia de usuario, además son de fácil instalación y mantenimiento.

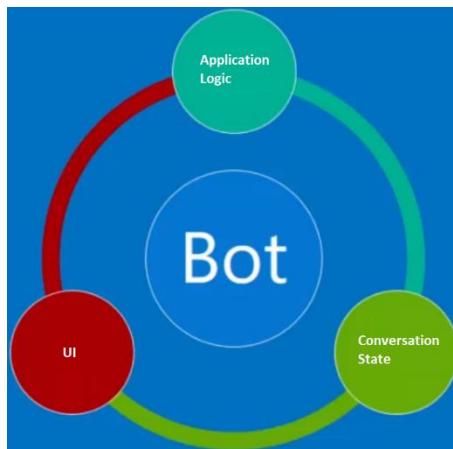


Ilustración 1.- Partes de un Bot conversacional

Como indica la Ilustración 1.- Partes de un Bot conversacional un bot conversacional está compuesto, a groso modo, de tres partes que unidas dan coherencia a la conversación:

1. *Application Logic*: recoge la lógica que utilizará el Bot para tratar la información enviada por el usuario.
2. *UI*: es la interfaz de usuario que da la cara al usuario.
3. *Conversation State*: es necesario un estado conversacional para saber cuál es el hilo de la conversación: de que se está hablando, cual fue la conversación anterior, cual es el marco de la conversación, etc.

2.1.2. Plataformas. Administración y funcionamiento

Para la creación de un Bot conversacional existen diferentes alternativas en el mercado como Microsoft Azure, IBM Watson, Amazon,... Se va a elegir la plataforma de **Microsoft, Microsoft Azure**, por aporta unas funcionalidades muy interesantes, dentro del marco gratuito de la aplicación, que permiten construir un Bot ampliamente, es decir, con las funcionalidades gratuitas de la plataforma se puede construir de sobra un Bot avanzado.

Información: Microsoft Azure es una plataforma increíblemente amplia que NO solo se dedica a la creación de Bots (Azure Bot Service) sino a muchas otras cosas, como servicio Web (App Service), máquinas virtuales, funciones simples sin servidor (Azure Functions), bases de datos (SQL Database o Azure Cosmos DB), equilibradores de carga para aplicaciones Web, monitores, grupos de recursos, cuentas de almacenamiento, y un largo etcétera.

Azure Bot Service

Microsoft Azure centra todos los servicios sobre Bots en AZURE BOT SERVICE que junto con otras aplicaciones de Azure consolidan un servicio completo. La versión 4.2 de AZURE BOT SERVICE es la que se utiliza para la creación del Bot, dando un gran paso frente a su anterior versión (3.0).

Esta plataforma cuenta con una serie de herramientas que se pueden resumir en tres conceptos básicos:

1. **Desarrollo acelerado.** AZURE BOT SERVICE proporciona una serie de herramientas de construcción, para el lenguaje de programación (C# o Node.js), y unas plantillas que van cambiando con el paso de las versiones. La herramienta para la construcción de código es MICROSOFT BOT FRAMEWORK SDK. La plantilla que se ha utilizado para la construcción del Bot ha sido la *Plantilla de Empresa (Enterprise Template)*.
2. **Aporte de inteligencia al Bot.** Los Bots no son simplemente conversadores, sino que también pueden hacer otras muchas cosas como reconocimiento facial, realización de recomendaciones inteligentes, traducción de idiomas, moderación del lenguaje y mucho más. Por lo tanto, los *Servicios Cognitivos* de Azure, llamado *Cognitive Services*, permite a los Bots ver, oír e interpretar de un modo más humano.
3. **Interacción con el usuario en cualquier lugar.** Al ser o ir de la mano de una aplicación Web, los Bots mediante internet pueden viajar a cualquier sitio del mundo actual. Dicha interacción se hace por medio de lo que Microsoft denomina *canales*, que forman parte del conector (servicio de conexión con el usuario, *Microsoft Bot Connector*). Los *canales* no son más que la plataforma final con la que el bot puede interactuar con el usuario. Entre ellos se encuentran Office365, Facebook Messenger, Skype, Cortana, Telegram, etc.

Importante: El Bot, que se ha querido construir, se ha enfocado de tal forma que prescinde de los servicios de pago de Microsoft Azure, por lo tanto, el punto 3 correspondiente al lanzamiento en la nube no se ha utilizado. En su lugar, se ha apostado por un lanzamiento local de pruebas donde se ha desarrollado todo el funcionamiento para, en un futuro visualizar la viabilidad del lanzamiento en la nube. Resumiendo, se han hecho uso tanto del primer punto como de parte del segundo.

Microsoft Bot Framework

MICROSOFT BOT FRAMEWORK es un conjunto de herramientas que Microsoft pone al servicio del usuario para construir, probar, proveer y gestionar cualquier Bot. Su uso permite crear Bots muy fácilmente, hacerles pruebas, administrarlos y conectarlos a *Servicios Cognitivos*, servicios de almacenamiento, servicios de telemetría, canales, traductores, moderadores del lenguaje, además de muchos otros aspectos.

Un framework...

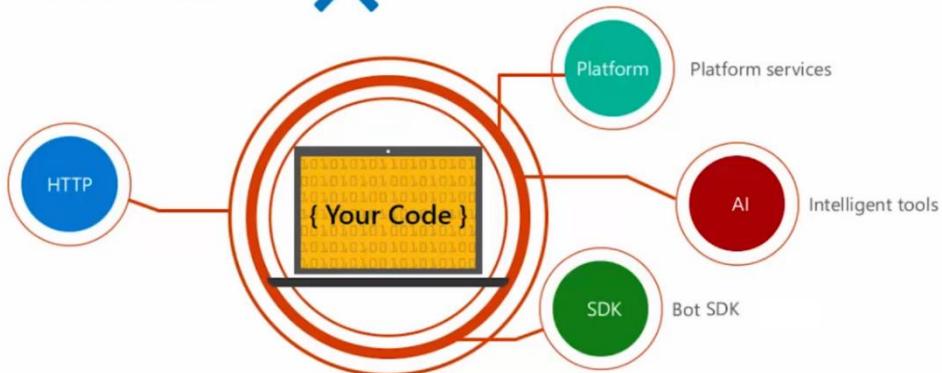


Ilustración 2.- Microsoft Bot Framework

A grosso modo, MICROSOFT BOT FRAMEWORK está compuesto por:

- La plataforma. Hace de cohesionador de las demás partes.
- Herramientas de inteligencia artificial. [LUIS](#), QnA, moderador, traductores, Visión, Voz, ...
- Herramientas para el desarrollo de código. Denominadas MICROSOFT BOT FRAMEWORK SDK se encargan de la creación, compilación y depuración del código.
- Herramientas de comunicación HTTP. Para enviar y recibir mensajes.
- Adicionalmente proporciona soporte para la emulación de Bots. El emulador de Bots de Microsoft se denomina **Bot Framework Emulator** y se usará en su versión 4 (V4) correspondiente a AZURE BOT SERVICE 4.2.

2.1.3. Experiencia conversacional

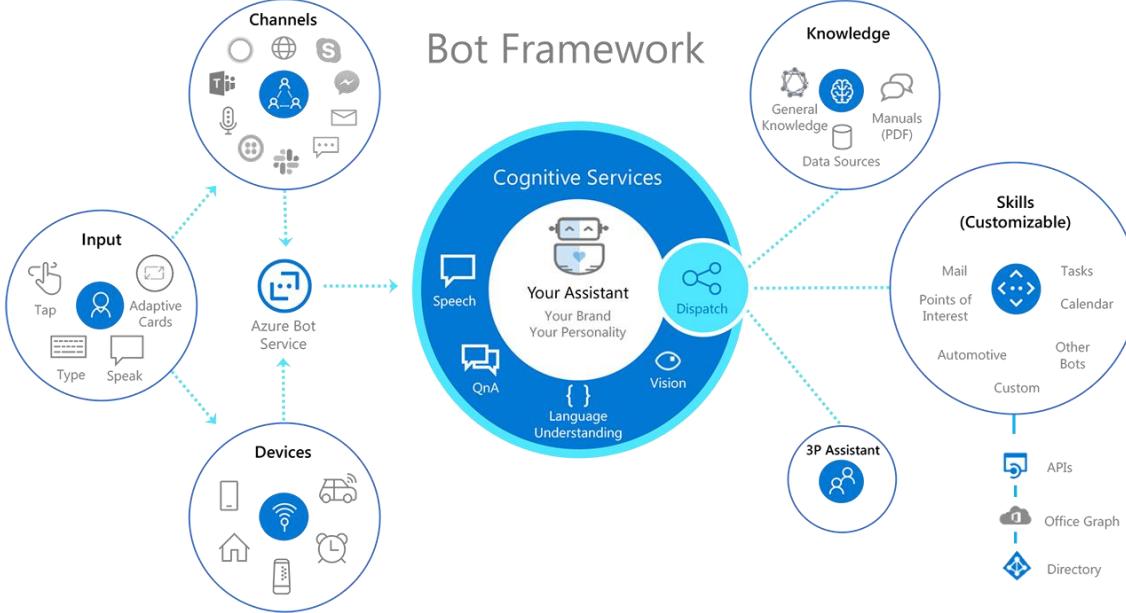


Ilustración 3.- Experiencia conversacional

La Ilustración 3.- Experiencia conversacional muestra las partes que intervienen en una experiencia conversacional completa desde la entrada del mensaje hasta la búsqueda de la información solicitada o el uso de habilidades. Las partes son:

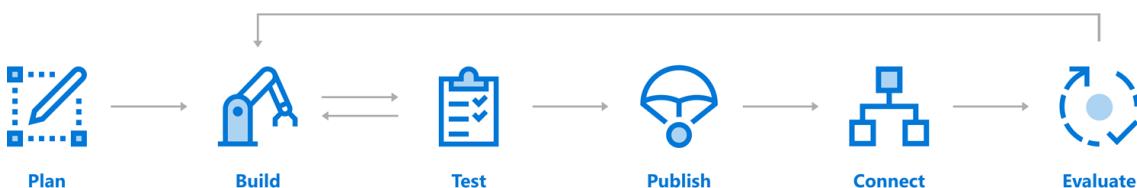
- **Entrada.** Los métodos de entrada disponibles son mediante tipo, gesticulación, botones y habla.
- **Canales o dispositivos.** Los canales y los dispositivos sirven de interfaz de entrada/salida de mensajes y se enlazan al Bot mediante el **BOT FRAMEWORK**. Los canales se sitúan, más bien, en el ámbito del PC y son Skype, Facebook Messenger, Cortana, ... Los dispositivos pueden ser móviles, coches, casas, etc inteligentes. El **Bot Framework Emulator** sirve de interfaz de entrada/salida de mensajes, pero solo se usa para el desarrollo y las pruebas.
- **Aplicación Bot.** Todo lo que comprende al Bot, es decir, su identidad y desempeño junto a los **Servicios Cognitivos (LUIS)**, QnA, Voz y Visión y modelos de distribución, computan el mensaje de entrada y realizan las acciones pertinentes.
- **Bases de conocimiento y habilidades.** Una vez procesado el mensaje de entrada se buscará la información pertinente en el almacenamiento del Bot o en bases de conocimiento externas; o se lanzarán una serie de habilidades/aptitudes como acciones para e-mail, acciones del calendario, señalar puntos de interés o lanzar una habilidad personalizada.

Importante: El Bot que se ha querido crear solamente recoge ciertas partes de la experiencia conversacional descrita anteriormente. A groso modo, la entrada del mensaje se recoge por Bot Framework Emulator a través del Bot Framework. El Bot, con la ayuda del

reconocimiento del lenguaje natural (**Luis**) y un modelo distribuidor, busca la información en el almacenamiento y la muestra.

2.1.4. Recorrido

La documentación de **AZURE BOT SERVICE** describe el desarrollo de un Bot, desde que se empieza a generar una idea hasta que se lanza al exterior. Según este recorrido, existen unas cuantas fases que detallan los puntos clave a seguir y que todo Bot atraviesa. Como se muestra en la imagen Ilustración 4.- Fases de desarrollo de un Bot, las diferentes fases son:



Planteamiento

Creación o construcción de la idea base partiendo de unos conocimientos amplios sobre Bots, sumado a buenas prácticas de programación, a las posibles necesidades del usuario y a la orientación del Bot hacia una comunicación clara y entretenida.

Lo más aconsejable es partir de un Bot simple de comunicación secuencial hasta desarrollar un Bot sofisticado con comprensión de lenguaje natural y comunicación compleja utilizando los servicios externos que se necesiten si es posible.

Construcción

La construcción se hará convirtiendo la idea en código ejecutable, para este cometido existen dos alternativas de código que son **C#** y **Node.js**. Aquí es donde reside la creación, desarrollo y depuración de código que va de la mano del **BOT FRAMEWORK SDK**.

Los servicios externos y herramientas, que proporciona **AZURE BOT SERVICE**, se encargan de complementar el código escrito y son de mucha utilidad. A continuación, se da una breve explicación de algunos de ellos:

- **Servicio inteligente de comprensión del lenguaje o Luis.** Servicio cognitivo que utiliza el procesamiento de lenguaje para reconocer intenciones, corregir errores ortográficos, etc. El reconocimiento de intenciones puede ser un gran aliado para la escritura de un código más simple e inteligente.
- **Preguntas frecuentes o QnA.** Servicio cognitivo que utiliza un almacén de preguntas-respuestas para contestar a las preguntas más frecuentes. Servicio exclusivamente de pago.
- **Administración de modelos cognitivos.** Un distribuidor de modelos cognitivos, para modelos **Luis** o QnA, determina de manera inteligente cuando usar un modelo u otro según el mensaje escrito por el Usuario.

- **Agregado de tarjetas o botones.** Mejora sustancial de la experiencia de usuario incluyendo texto con énfasis, viñetas, imágenes, videos, enlaces, botones, ... y distribución de todos ellos de manera cuadriculada (filas, columnas, etc).

Prueba

La prueba de funcionamiento se realizará paulatinamente durante la construcción del Bot, además habrá una prueba final antes de su lanzamiento que comprobará el correcto funcionamiento de las partes que lo componen. Las pruebas se realizan con el depurador de la aplicación y el emulador en conjunto. Se recomienda usar el depurador paso a paso para comprobar los fallos de funcionamiento complejos (la forma más efectiva será el ensayo y error, ¡eso sí! partiendo de una base de conocimientos).

Bot Framework Emulator

El emulador, **Bot Framework Emulator V4**, muestra la parte visual del Bot, siendo una herramienta imprescindible para la prueba de Bots. Muestra tres pantallas según se ve en Ilustración 5.- Interfaz Bot Framework Emulator V4:

- **Principal - Conversación.** Muestra la conversación entre el Usuario y el Bot.
- **Segunda - Inspector.** Muestra los datos de cada mensaje enviado y recibido (en formato *.json*), no solo el texto sino el canal utilizado, cuando se envió el mensaje, localización, la dirección de hospedaje, las acciones, las intenciones, las entidades, y un largo etcétera, según el tipo de mensaje.
- **Tercera - Registro.** Muestra el historial de la conversación, tanto los mensajes de entrada como los de salida. Los mensajes se muestran en protocolo HTTP.

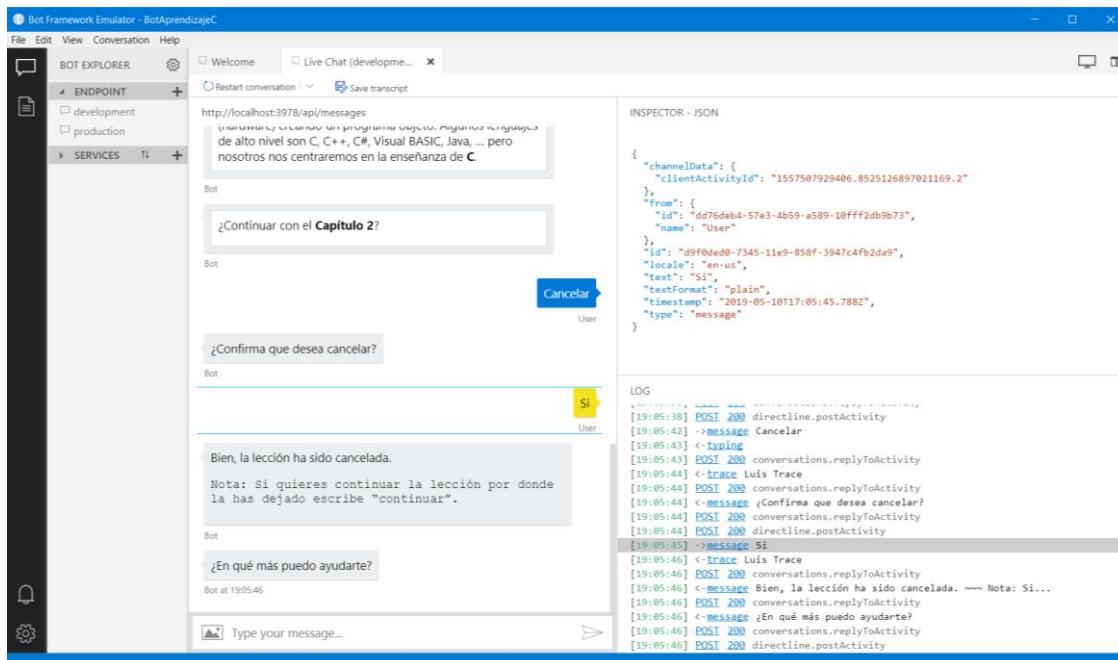


Ilustración 5.- Interfaz Bot Framework Emulator V4

Publicar

Una vez el Bot está construido en su totalidad (cumple con las ideas de diseño, sigue unos patrones de sencillez y entretenimiento, y cumple con los objetivos del usuario) estará listo para el lanzamiento en la nube en la plataforma de Azure o en nuestro propio sitio Web.

El cometido de un **Bot conversacional** es que lo utilicen el mayor número de personas, así, la forma más sencilla de conseguirlo es lanzarlo en la Web.

Conectar

Si el Bot es lanzado en la plataforma de Azure este podrá ser provisto de una serie de canales (puntos de conexión con el usuario) que aumentarán la posibilidad de que un nuevo usuario se conecte al Bot, creando un sistema de comunicación multiplataforma rápido y asequible. Entre los canales disponibles se encuentran Facebook, Messenger, Skype, Telegram, Cortana, entre otros.

Evaluación

Mediante el recuento de datos que proporciona la plataforma de Azure, y tras la activación del servicio *Application Insights*, se pueden recopilar datos de telemetría como los canales utilizados, el tráfico de mensajes entrante, la información buscada, las opciones elegidas, el tiempo usado en cada apartado, etc. Esto nos dará una idea de que canales son los más adecuados, si hay que cambiar la información que proporciona el Bot, si el Bot está teniendo tirón, y demás cosas.

Importante: *Las fases recorridas por el Bot, que se ha querido construir, son las de planteamiento, construcción y prueba. Las tres restantes se han excluido por que, el objetivo del proyecto es la creación del Bot no el lanzamiento en la nube.*

2.2. Diseño del Bot en Azure

2.2.1. Funcionamiento de una Aplicación Bot

Importante: Toda la información explicada en este punto, Funcionamiento de una Aplicación Bot, se ha extraído de la documentación de Azure Bot Service de Microsoft y se ha explicado de manera diferente (más sencilla). La referencia a dicha documentación se sitúa en la Agradecimientos concretamente en Aprendizaje sobre el Bot.

El lenguaje de programación utilizado para la construcción del Bot ha sido **C#** debido a que, la documentación que Microsoft pone a disposición del consumidor está prácticamente en su totalidad en dicho lenguaje.

Nota: También se da un gran soporte al lenguaje de programación JavaScript, pero resulta menos completo que el soporte para C#.

A continuación, se va a profundizar un poco más en el funcionamiento, estructura y diseño de un Bot... todo ello visto desde el punto de vista de la codificación en **C#**.

Funcionamiento básico

Una aplicación Bot es como una aplicación Web tradicional con la peculiaridad de que la interacción Usuario-Aplicación se hace de forma conversacional mediante preguntas y respuestas.

Cada una de las interacciones, a grosso modo, genera una *actividad* que almacena el texto del mensaje, las acciones, el tipo de actividad, el canal, la localización, ... Estas interacciones se realizan por medio del **BOT FRAMEWORK** que conecta el canal (Usuario – mediante Skype, Facebook, ...) con el Bot.



Tipos de Actividad

Por norma general se distinguen dos tipos de **actividad**:

- Actividad – *Actualización de la Conversación*: se suele dar al comienzo de la conversación, cuando algo o alguien se une a la conversación, por ejemplo, el canal o el Bot. Se utiliza para que el Bot envíe un primer mensaje como un mensaje de bienvenida al Usuario o cualquier otro mensaje que se precise.
- Actividad – *Mensaje*: cada interacción, que el Usuario hace con el Bot y viceversa, genera una actividad mensaje. Cuando se recibe una “actividad de mensaje entrante” del Usuario, el Bot actúa en consecuencia desencadenando las acciones pertinentes y generando una “actividad de mensaje saliente”. Tratándose de un Bot conversacional se puede decir que es la entrada por excelencia y la que accede a la lógica principal del Bot.

Nota: Se podrá tanto responder con otro mensaje (video, imagen, texto hablado, texto plano, texto con marcaje, tarjetas enriquecidas, botones de acción, etc) como utilizar una habilidad (poner un temporizador, enviar/eliminar/responder e-mails, crear/eliminar tarea del calendario, etc).

¿Qué es un turno de conversación?

En una conversación humana, los integrantes hablan por turnos. Por ejemplo, en el primer turno, el emisor envía un mensaje que recibe el receptor y, en el segundo turno, el receptor responde al emisor con otro mensaje; y así sucesivamente.

En resumen, un turno de conversación se define como el periodo desde que se recibe un mensaje hasta que se emite una respuesta.

Para Bots, un turno de conversación sería tanto el periodo desde que el Bot recibe el mensaje de entrada hasta que produce una contestación, como el periodo desde que el que el Usuario recibe el mensaje de salida y emite un nuevo mensaje de entrada.

Contexto de turno

A su vez los humanos necesitan de un contexto o una situación específica, para comprender la conversación y mantener la coherencia en la información transmitida. No sería correcto que el emisor estuviese hablando de un tema y el emisor de otro diferente. Así, la arquitectura Bot se diseñó de manera similar en este aspecto, primero conversando por turnos y segundo manteniendo el contexto de la conversación.

El **contexto de turno** recoge la información sobre el estado del turno, el adaptador (control de errores, el software intermedio, ...), la actividad de entrada, ... Se encarga de llevar la información de entrada (Usuario → Bot) a todos los componentes del software intermedio (*middleware*) y a la lógica de la aplicación (desempeño del Bot), para más tarde, pasar la información de salida (Bot → Usuario) por el *middleware* y devolverla al Usuario.

Los conceptos de *actividad* y *contexto de turno* son independientes entre sí, pero el **adaptador** se encarga precisamente de introducir la actividad como parte del contexto del turno. El adaptador trabaja así... cuando recibe una actividad de entrada, crea un contexto de turno a medida para esa actividad y cuando se genera una actividad de salida, destruye el contexto anterior y crea otro para la actividad de salida.

Nota: Tanto la entrada como la salida de una actividad (y la arquitectura Bot en general) se ejecutan de forma asíncrona (await), principalmente para evitar la pérdida del contexto de turno. Si no se marcara así, el subproceso que regula el turno principal desecharía el objeto - contexto de turno - cuando terminase, impidiendo al resto del código utilizarlo.

El software intermedio o *middleware*

El **software intermedio o middleware** es, a grandes rasgos, una clase que se encuentra entre el canal (Usuario) y la lógica de la aplicación (desempeño del Bot) y contiene un conjunto de componentes que se ejecutan en serie dando a cada uno la posibilidad de actuar sobre la actividad.

El adaptador transportará la actividad de entrada (Usuario → Bot) hacia el software intermedio. La etapa final del *middleware* arrancará la función del controlador de turnos dando paso a la lógica de la aplicación (desempeño del Bot). Dicha lógica enviará una actividad de salida (Bot → Usuario), activando de nuevo el software intermedio.

El orden de los elementos del *middleware* juega un papel crucial en el procesamiento de la actividad, es decir, habrá que seguir el orden que mejor se adapte a las necesidades de la aplicación.

Algunos elementos del *middleware* pueden ser: el registro de cada actividad, el control de excepciones (errores), la traducción, el registro del historial de la conversación, el tipeo, la ubicación, el autoguardado de estados (Usuario y Conversación), etc.

Recorrido de un mensaje y partes involucradas

Todo el Funcionamiento básico, es decir, el recorrido de un mensaje y todas las partes que atraviesa, se recoge en Ilustración 6.- Gráfico de funcionamiento de un Bot.

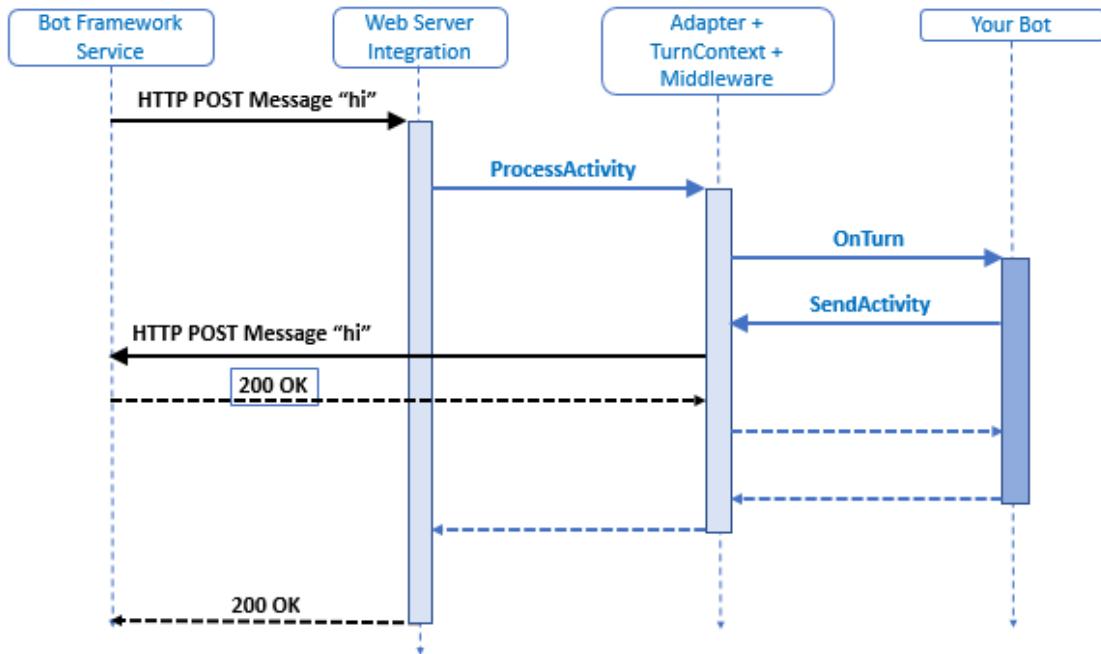


Ilustración 6.- Gráfico de funcionamiento de un Bot

Rápidamente:

- El BOT FRAMEWORK recoge el mensaje de entrada del Usuario como una actividad.
- Esa actividad es reconocida al instante por el controlador (Web Server Integration), específicamente por ser del tipo HTTP POST.
- HTTP POST desata el método *ProcessActivity* que ejecuta el adaptador. El adaptador genera el *contexto del turno*, le incrusta la actividad y lo envía al *middleware* que, mediante el método controlador de turnos (*OnTurnAsync*), lo va delegando a todas sus herramientas. La última herramienta del *middleware* delega el funcionamiento al método controlador de turnos del Bot dando paso a la lógica del Bot (desempeño del Bot).
- El Bot procesa la actividad (incrustada en el *contexto del turno*) y genera el mensaje de salida. El mensaje de salida genera la nueva actividad que desata el método *SendActivity* enviándola al adaptador.
- El adaptador elimina el *contexto del turno* anterior y genera uno nuevo, al cual le incrusta la nueva actividad, y lo envía al *middleware*. El *middleware* pasa de nuevo el *contexto del turno* por sus herramientas mediante delegación y envía el mensaje al usuario a través del BOT FRAMEWORK.
- Si el mensaje de salida ha llegado correctamente al Usuario, se envía un mensaje verificador, 200 OK, que le dice al adaptador y posteriormente al Bot que todo ha salido bien.
- Si se acepta el mensaje verificador anterior, se envía otro mensaje verificador, 200 OK, diciéndole a todas las partes involucradas que el conjunto “mensaje entrada-salida” ha sido procesado correctamente.

Estructura del Bot

Puesta en marcha de la Aplicación Bot

Un Bot es una adaptación de una *aplicación Web* que, a su vez, sigue un paradigma de programación controlado por eventos, delegados, tareas multihilo y espera asíncrona.

Siguiendo el paradigma de Microsoft en aplicaciones Web y el lenguaje de programación C#, la puesta en marcha se realiza mediante los archivos **Program.cs** y **Startup.cs**.

- **Program.cs** se encarga de hospedar, construir y correr la aplicación, además de llamar a Startup.cs.
- **Startup.cs** llama a *ConfigureServices* para cargar los servicios utilizados por el Bot y el propio Bot (nuestrobot), además de las herramientas del *middleware*, la detección y registro de errores que se puedan producir, la configuración del enlace con el host local o la Web App Bot y la creación/cargar del estado de la Conversación y del Usuario. También llama a *Configure* para especificar que la aplicación usará BOT FRAMEWORK, es decir, que usará Bots y todo su marco de bibliotecas.

Una vez cargados satisfactoriamente ambos archivos, el Bot se pone en funcionamiento a la espera de una actividad de entrada.

Ahora el archivo, nuestrobot.cs, que incluye la función **OnTurnAsync** (heredada la interfaz *IBot*) recibirá la actividad de entrada. Este archivo constituye el punto de entrada a la lógica del Bot y será ejecutado para cada actividad de entrada.

Archivo .bot

El archivo **.bot** contiene los IDs y Claves de los puntos de conexión (local o Web) y los servicios utilizados (**Luis**, Dispatch, QnA, moderador de contenido, telemetría, transcripciones, traducción, voz, etc.).

Un ejemplo... si el Bot estuviese corriendo en la nube, tuviese reconocimiento de lenguaje y almacenamiento del estado de la Conversación y del Usuario, necesitaría IDs y Claves del: punto de conexión Web, servicio de reconocimiento de lenguaje y servicio de almacenamiento. Particularmente en la plataforma de Microsoft Azure, utilizaría Web App Bot, **Luis** y CosmosDB.

Archivo appsettings.json

En revisiones antiguas de AZURE BOT SERVICE, **appsettings.json** contenía toda la información que ahora contiene el archivo **.bot**. En este momento apenas contiene la ruta de acceso al archivo **.bot** y la localización por defecto (es-ES).

Administración de estados

Una aplicación Web o, en este caso, un Bot, carecen de estado. Un Bot no necesita de un estado para funcionar, pero para un Bot conversacional es casi imprescindible. Es necesario mantener una coherencia en la conversación para no responder sin sentido a una pregunta. Por ejemplo, saber quién ha enviado el mensaje para saber a quién responder o responder a un mensaje de forma correcta.

Almacenamiento

Un **almacenamiento** se usa principalmente para guardar los estados de cada Usuario que se conecte al Bot. Esta funcionalidad se completa con el servicio de autenticación, es decir, los estados se guardarán en referencia al Usuario autenticado.

Los únicos almacenamientos persistentes disponibles para el estado del Bot pertenecen a Microsoft o a terceros. Ambos son servicios en la nube y están incluidos en la plataforma Azure de Microsoft, además son servicios de pago. Dos ejemplos son: *Azure Blob Storage* y *Azure Cosmos DB*.

La única opción gratuita es el almacenamiento en memoria, *MemoryStorage*, incluida en **BOT FRAMEWORK**, pero proporciona almacenamiento temporal hasta que la aplicación Bot se cierre.

Propiedades de estado

El estado de Bot se almacena como **propiedades de estado** y existen tres incluidas en el **BOT FRAMEWORK**: *estado del Usuario*, *estado de la Conversación* y *estado de la Conversación privada*. Estas tres propiedades de estado se derivan de la clase estado del Bot, *BotState*.

- El *estado del Usuario* se usa en momentos puntuales y se encarga principalmente de almacenar los datos más relevantes como por ejemplo su nombre, las consultas que hizo, las preferencias, el historial, los datos indirectos, etc.
- El *estado de la Conversación* es utilizado principalmente para fijar el diálogo base del Bot del que derivan todos los demás. Mantiene un seguimiento de la conversación para por ejemplo saber dónde se dejó o que fue lo último consultado. A parte, mantiene la coherencia en cada turno de conversación por ejemplo para responder correctamente a una pregunta.
- El *estado de la Conversación privada* se limita a conversaciones de grupo donde, para cada Usuario del grupo, es necesario llevar un seguimiento específico.

Si se usasen varios almacenamientos en la misma aplicación Bot, habría que crear las propiedades de estado correspondientes para cumplir con el funcionamiento deseado.

Descriptoros de acceso

Un **descriptor de acceso** se obtiene a través de una propiedad de estado (principalmente del Usuario) y de una clase que almacena un tipo concreto de estados, es decir, permite acceder a unos estados concretos de una propiedad de estado. Por ejemplo, para acceder a una clase que almacena unos estados concretos del Usuario (nombre, localización, dirección de correo, ...) será necesario crear un descriptor de acceso a partir del estado del Usuario y de esa clase concreta.

Nota: Se pueden crear cualquier cantidad de clases de estado diferentes para almacenar cualquier tipo de datos, eso sí, es conveniente que estén bien delimitadas.

El uso principal de un descriptor de acceso, *accessor*, es el de leer o escribir en los estados correspondientes de una propiedad de estado.

Almacenamiento manual

Para poder modificar, de manera manual, un grupo de estados concreto de una propiedad de estado habrá que crear un descriptor de acceso a esa propiedad junto a la clase que contiene el grupo de estados.

El método *GetAsync* permite obtener la propiedad de estado a modificar. Una vez modificado, se fija con el método *SetAsync*. Como el cambio solamente se ha hecho de forma local, habrá que utilizar el método *SaveChangesAsync* para conseguir un cambio en el almacenamiento.

Almacenamiento automático

Existe otra forma de guardar los cambios en el almacenamiento. Esta forma se basa en el autoguardado por -turno de conversación-, o lo que es lo mismo, en el *middleware* (el *middleware* se activa en cada turno de conversación). Mediante la adición de la herramienta *AutoSaveStateMiddleware* al *middleware*, se consigue guardar una o varias propiedades de estado.

Biblioteca de diálogos

La **biblioteca de diálogos**, *Dialog*, se encarga de la conversación con el Usuario y es considerada la parte central del Bot donde reside la verdadera naturaleza de un Bot conversacional. Cada diálogo está diseñado para cumplir con un desempeño en concreto y pueden ser desencadenados a través del código del Bot, mediante un mensaje de entrada recibido del Usuario.

Tipos de diálogos

Existen tres tipos de diálogos diferentes dentro del **BOT FRAMEWORK**: *diálogo de petición* (*PromptDialog*), *diálogo en cascada* (*WaterfallDialog*) y *diálogo de componentes* (*ComponentDialog*).

- El diálogo de petición se encarga de pedir información al Usuario en forma de elección, texto, número, fecha, etc. El método peticionador (*PromptAsync*) es el encargado de realizar la petición al Usuario y contendrá un mensaje sumado al diálogo de petición; primero realiza la solicitud al Usuario y segundo, si la respuesta es válida se devuelve el valor y, si no se vuelve a lanzar el peticionador. Además del validador predeterminado que incluye cada tipo de diálogo de petición, se puede crear uno personalizado.

Existen diálogos de petición de...

- Archivos adjuntos. Sigue el mismo principio que el diálogo de petición, pero devolverá una colección de todos los archivos adjuntos.
- Elección. Sigue el mismo principio que el diálogo de petición, pero devolverá el objeto seleccionado.
- Confirmación. Sigue el mismo principio que el diálogo de petición, pero devolverá un valor booleano.
- Fecha y hora. Pide una fecha y devolverá un objeto de tipo fecha.
- Número. Sigue el mismo principio que el diálogo de petición, pero devolverá un valor numérico.
- Texto. Pide una entrada de texto y devolverá una cadena.

Los valores devueltos serán utilizados por el Bot para desencadenar un código u otro.

- El diálogo en cascada se encarga de mantener una conversación paso a paso de forma guiada en la que los mensajes que envíe el Usuario determinarán la ruta que seguirá la

cascada. En realidad, un diálogo en cascada se complementa con métodos peticionadores, por ejemplo, se le pedirá al Usuario que elija la opción que más le convenga para, en el siguiente paso de cascada, actuar en consecuencia.

- El diálogo de componentes se encarga de agrupar diálogos en conjuntos de diálogos. Puede contener diálogos en cascada o diálogos de petición. Permite crear diálogos independientes para tratar escenarios concretos y puede dividir un conjunto de diálogos grande en diálogos más pequeños, es decir, es un diálogo de diálogos. Cada diálogo agregado se nombra con un identificador diferente y cuando sea necesario utilizarlo se llamará por su identificador.

Nota: La relación que sigue cada tipo de dialogo... un diálogo de componentes definirá un tema a tratar y estará compuesto de los diálogos en cascada que definen cada punto de ese tema. A su vez, cada diálogo en cascada contendrá los diálogos de petición correspondientes que ayudarán a resolver dicho punto.

Elementos de la biblioteca de diálogos

Los elementos principales de la biblioteca de diálogos son el *conjunto de diálogos*, el *contexto del diálogo* y el *resultado del diálogo*.

- El *conjunto de diálogos* se recoge en el diálogo de componentes (*ComponentDialog*). Es posible crear conjuntos de diálogos para múltiples tareas. Es especialmente útil cuando quieras aumentar la funcionalidad de un Bot que se dedica a una sola temática.
- El contexto del diálogo (*DialogContext*), se crea a partir del contexto de turno y del estado diálogo, *DialogState* (que forma parte del estado de la Conversación) y es, en definitiva, el que mantiene la coherencia en la conversación. El contexto del diálogo permite: iniciar (*BeginDialogAsync*), reemplazar (*ReplaceDialogAsync*), repetir (*RepromptDialogAsync*), continuar (*ContinueDialogAsync*) y finalizar un diálogo (*EndDialogAsync*) o todos los diálogos (*CancelAllDialogsAsync*).
- En ocasiones es útil recibir el *resultado del diálogo* para, por ejemplo, almacenar datos, saber si el diálogo se ha realizado correctamente, actuar en consecuencia desatando un código determinado, etc.

Pila de diálogos

La pila de diálogos de cualquier Bot se inicia, de alguna manera, por la función controladora de turnos a la lógica del Bot también llamada punto de entrada de mensajería (*OnTurnAsync*).

Más concretamente, mediante la llamada al método -comenzar diálogo (*BeginDialogAsync*)- del contexto del diálogo (*DialogContext*).

Por ejemplo, mediante un mensaje de entrada del Usuario, el código insertará el diálogo base considerándose el diálogo activo en este momento. Ahora, se pueden insertar otros diálogos (en la parte superior) en la pila sin problemas. Si es así, el último diálogo insertado será diálogo activo. Cuando termine su ejecución (llamada al método *EndDialogAsync*) será sacado de la pila de diálogos siendo, ahora, el diálogo activo el inmediatamente inferior. Cuando todos los diálogos en la pila hayan finalizado, el Bot esperará otro mensaje de entrada para comenzar con una nueva conversación.

Enfoque y diseño

Aspectos clave de una conversación Usuario-Bot

Un Bot debe seguir un diseño, tal que, la conversación con el usuario sea lo más natural posible, es decir, que su comportamiento sea lo más parecido a una conversación humana (comunicación clara y concisa, cambio de temática, interrupciones, etc).

**Un Bot conversacional debe cumplir con su cometido, pero siempre
respetando o priorizando la experiencia de usuario.**

Hay que evitar en todo momento nublar la experiencia de usuario, por ejemplo, pidiendo repetitivamente contestar a una pregunta sin opción a no hacerlo, mostrar información diferente a la que se pregunta, no responder a una pregunta (Bot colgado), responder con aspectos obvios que no aportan nada, aportar información innecesaria de conversaciones anteriores, etc.

Flujo de procedimientos

Cada tipo de Bot tendrá unas formas de proceder u otras, lo que se denomina *flujo de procedimientos*. Por ejemplo, la representación de la información dependerá de la temática que siga el Bot, es decir, no es lo mismo la toma de pedidos que la gestión de correos de una empresa o la enseñanza de alguna información.

Interrupciones

Otro punto que destacar es el cambio en la conversación o *interrupciones*, por ejemplo, para consultar otra información, reiniciar la conversación activa, consultar la ayuda, etc. Las interrupciones sirven principalmente para interrumpir o cancelar la conversación activa. Todo Bot conversacional debe poseer un control de interrupciones para conseguir la mejor experiencia de usuario posible.

Una interrupción debe ser tratada de la mejor forma posible. *¿Cómo debería responder el Bot a una interrupción? - Insistiendo en terminar el tema tratado para después contestar al nuevo, eliminar el tema tratado para comenzar con el nuevo o intentar dar respuesta al nuevo tema y después continuar con el tema actual -* Como ocurre con el flujo de procedimientos no está claro cuál será la mejor opción y esta dependerá del desempeño del Bot.

Aspectos que influyen en la experiencia de usuario

Un Bot conversacional no es simplemente un conversador a través de texto, sino que puede incorporar otros aspectos parecidos a sitios Web como botones de acción, tarjetas con información múltiple (video, audio, botones, texto), ... además pueden utilizar herramientas de compresión del lenguaje e incluso entrada y salida de voz.

La mayoría de las respuestas en texto que proporciona un Bot, admiten el marcado MARKDOWN que da la opción utilizar títulos, subtítulos, viñetas, negrita, cursiva, citas, ... en el texto de forma parecida a un procesador de textos.

Controles de usuario

Los *controles de usuario* -como botones, imágenes, carruseles, menús, etc- se encargan en la mayoría de las ocasiones de agilizar la conversación y es una manera excelente de limitar las posibles respuestas.

En el aspecto más general del Bot, de cara a la interfaz de usuario, los controles poseen un doble beneficio, a la vez que se consigue una mejor experiencia visual y estructural también se consigue llevar la temática por el camino elegido.

Tarjetas enriquecidas

Las *tarjetas enriquecidas* son un grupo de tarjetas que permiten crear experiencias de usuario diversas y adaptadas a cada modalidad de Bot. Agrupan la información en un marco permitiendo incluir texto, imágenes, videos, elecciones, etc según el tipo de tarjeta. Entre ellas se encuentran las tarjetas adaptativas, de animación, de audio, de ídolo, de miniaturas, de recibo, de inicio de sesión, de acción sugerida y de video. Es posible mostrar un grupo de tarjetas mediante un carrusel o una lista.

- Las *tarjetas adaptativas* o tarjetas adaptables son uno de los aspectos más importantes para una buena experiencia de usuario. Permiten mezclar imágenes, texto, audio, videos, botones, enlaces Web, etc ordenados de la forma que mejor se adapte al cometido del Bot (columnas, bloques, alineación, formatos de texto, *HTML*, *MARKDOWN*, etc). Todas estas opciones dejan un margen de maniobra perfecto para crear un gran abanico de tarjetas.

Importante: Según el canal que utilice el usuario para comunicarse con el Bot, la información que muestra una tarjeta puede ser ligeramente diferente. Por ejemplo, se debe mostrar una tarjeta con una serie de botones, pero el canal no admite esa característica y, en vez de eso, se muestra una imagen plana. Por desgracia las tarjetas adaptativas podrían ser las más perjudicadas en este aspecto.

Compresión del lenguaje

La *compresión del lenguaje*, como su propio nombre indica, se encarga de comprender el texto del mensaje de entrada (Usuario) y actuar en consecuencia. Concretamente analizan la información de entrada, compuesta de palabras clave, permitiendo identificar su intención para, más tarde, actuar sobre la lógica del Bot desatando una acción concreta. Las aplicaciones encargadas de este cometido son las que incorporan comportamiento cognitivo como *LUIS*, QnA, Bing Web Search, Computer Vision, voz, moderador de contenido, traductor, entre muchas otras.

Importante: Como se puede imaginar, estas aplicaciones no son la panacea y en algunos aspectos pueden no ser lo suficientemente eficientes. En caso de abarcar preguntas abiertas difíciles de procesar, se recomienda encauzar la pregunta mediante una serie de preguntas específicas. Con esto se conseguirá el mismo objetivo, pero se reducirá la complejidad del código del Bot.

2.2.2. BOTAPRENDIZAJEC. Funcionamiento general

En relación a la propuesta del proyecto y partiendo de la metodología de sencillez seguida, el nombre del Bot ha sido finalmente [BOTAPRENDIZAJEC](#).

A partir de aquí se va a mostrar la estructura, el funcionamiento, los servicios utilizados, la exposición del APRENDIZAJE DE C, los puntos clave, el contenido general y un largo etcétera de las partes que componen el Bot. El lenguaje de programación utilizado para diseñar el Bot ha sido [C#](#) y, por ende, la Herramientas de construcción utilizada ha sido [VISUAL STUDIO 2017 COMMUNITY](#).

BOTAPRENDIZAJEC se encarga de la enseñanza del lenguaje C en general, pero está más orientado a la enseñanza al alumnado de universidad.

La forma de exposición de la información es mezcla del formato libro y la enseñanza interactiva. Eso sí, siempre cuidando mostrar la información de forma clara (mediante una interfaz vistosa) y precisa (yendo al grano, pero sin descuidar ningún detalle) evitando en todo momento información excesiva (que pueda generar pesadez).

Hace un buen uso del marcaje MARKDOWN para dar énfasis a frases o palabras, incluir títulos, utilizar viñetas, ... Además de mostrar notas y avisos sobre aspectos relevantes del lenguaje C.

Posee un funcionamiento muy sencillo que hace la obtención de información por parte del Usuario muy amena. Sigue una estructura simple que contiene una tarjeta menú, una tarjeta ayuda, un grupo de tarjetas índice del temario e índice de los problemas y una tarjeta sobre la bibliografía.

La información sobre el APRENDIZAJE DE C esta partida en TEMARIO y PROBLEMAS. El TEMARIO recoge toda la enseñanza paso a paso dividida en capítulos y apartados por capítulo. Los PROBLEMAS recogen una serie de problemas variados unos explicados, otros con preguntas y otros propuestos.



Tarjeta de Introducción

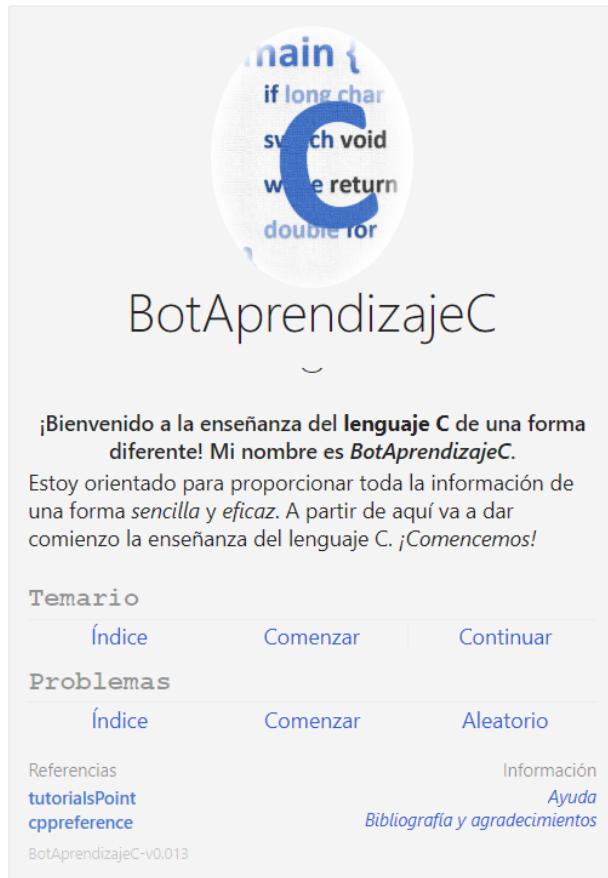


Ilustración 7.- Tarjeta de Introducción

Ilustración 7.- Tarjeta de Introducción:

La *Tarjeta de Introducción* muestra de un vistazo la intención del Bot proporcionando su nombre e icono, dando una explicación sencilla de su desempeño y permitiendo elegir alguna de las opciones recomendables (índice o inicio del TEMARIO, índice o inicio de los PROBLEMAS, ayuda, bibliografía y agradecimientos, problema aleatorio, entre otras), además proporciona enlaces externos de interés.

El desencadenamiento de la tarjeta se realiza cuando: hay una actualización de la conversación, *ConversationUpdate* (el Bot o el canal (Usuario) se unen a la conversación), y el Bot se une a la conversación. También es posible escribir “menu” o “tarjeta de introducción” para desencadenarla.

Información detallada

Recoge la IMAGEN (icono) y el NOMBRE (identificador) del Bot, un texto de bienvenida y un texto corto de su desempeño. Además, proporciona botones para un acceso rápido a las funciones más útiles como:

- TEMARIO.

- **Índice.** Muestra el índice del temario dividido en capítulos, apartados de capítulo, subapartados y ejercicios. Además, proporciona información sencilla de como buscar un apartado en concreto.
- **Comenzar.** Comienza el TEMARIO por el primer apartado del capítulo 1.
- **Continuar.** Continua por el último punto consultado del APRENDIZAJE DE C. En caso de no haber consultado uno se lanza un mensaje de error.
- **PROBLEMAS.**
 - **Índice.** Muestra el índice de los problemas dividido en capítulos y problemas por capítulo. Además, proporciona información sencilla sobre como buscar un problema.
 - **Comenzar.** Comienza los PROBLEMAS por el primer problema del capítulo 1.
 - **Aleatorio.** Lanza un problema aleatorio.
- **Ayuda.** Proporciona acceso a la *Tarjeta de Ayuda*.
- **Bibliografía y agradecimientos.** Muestra toda la información sobre la bibliografía utilizada para la realización del Bot y agradece a las partes implicadas, libros, páginas Web, ... su colaboración.
- **Enlace Web (tutorialsPoint).** Enlace Web que contiene referencias a C.
- **Enlace Web (cppreference).** Enlace Web que contiene referencias a C.

Escritura y pulsación

Las formas de interacción principal con el Bot son: el uso de **escritura por teclado** (mensajes) y el uso de **pulsación** (botones).

Nota: Hay que decir que, a nivel de código, el Bot está construido para soportar el lenguaje hablado (Voz), pero la forma en la que se ha querido mostrar la información, hace que este sistema sea contraproducente, aparte de ser un servicio de pago. En resumen, no se ha activado y no se puede utilizar.

A continuación, se muestra el funcionamiento de cada parte y su convivencia conjunta.

Mensajes entrantes

Un mensaje entrante se considera todo mensaje que el Usuario envíe al Bot (Usuario -> Bot) y se compone simplemente de la consulta escrita que el usuario hace al Bot. Estas consultas hacen que el Bot actúe en consecuencia y desencadene la información por medio de código.

Cuando la información mostrada está relacionada con el APRENDIZAJE DE C, el Bot automáticamente propone al usuario diferentes opciones entre las que se encuentran: continuar al siguiente apartado, cancelar la lección, información al siguiente apartado o mostrar solución a un ejercicio.

Mensajes salientes

Un mensaje saliente se considera todo mensaje que el Bot envíe al Usuario (Bot -> Usuario) y pueden ser mensajes simples o mensajes elaborados.

Los mensajes salientes elaborados serán en su gran mayoría los derivados del APRENDIZAJE DE C y seguirán un formato de claridad parecido en la información mostrada. En primer lugar, la

información mostrada es eficaz, es decir, se basa en un buen resumen y explicación tanto visual como escrita y, en segundo lugar, el marcaje MARKDOWN enfatiza las partes más relevantes, les da cohesión y distinción; consiguiendo que el usuario coja la idea de una pasada. Cada punto del APRENDIZAJE DE C será incrustado en *tarjetas adaptativas* y junto a imágenes, notas, avisos, restricciones y URLs, se conseguirá una enseñanza completa.

El resto de los mensajes, mensajes salientes simples, en ocasiones contienen también escritura MARKDOWN pero carecen de información visual. Se componen principalmente de texto corto y se utilizan como *mensajes de control*, que el Bot utiliza para avisos, cancelación, cierre de apartado, etc.

Sugerencias

Una sugerencia es considerada toda información insertada como bloque en algunas de las tarjetas del APRENDIZAJE DE C y se encargan de avisar, notificar o restringir algunas acciones del lenguaje C.

- **Notas.** Información sugerida que se nutre de las buenas prácticas de C. Se representan mediante énfasis en un recuadro de color de fondo azul claro.
- **Avisos.** Advertencia sobre los errores comunes de C. Se representan mediante énfasis en un recuadro de color de fondo azul oscuro.
- **Restricciones.** Limita el uso de alguna funcionalidad de C en determinados problemas de los PROBLEMAS. Se representan mediante énfasis en un recuadro de color de fondo naranja claro.

Botones

Se insertan o van incluidos en *tarjetas adaptativas*. Hay diferentes opciones en el tipo de acción desencadenada al pulsar el botón. Las usadas por el Bot son para:

- Abrir una URL (OpenUrl): abre una URL en el navegador web predeterminado.
- Enviar un mensaje (Submit): envía la información que contiene el botón elegido por el Usuario, al Bot. Se han creado dos tipos de botones diferentes para distinguir entre botones que desencadenan eventos y botones que simplemente contienen texto:
 - **Botones especiales:** Están compuesto por “cadena de texto + Clave”. Sirven para realizar acciones específicas como desencadenar eventos creados, por ejemplo, acciones derivadas de la *Tarjeta de Introducción* como indiceTEMARIO*IntroEvent* o de la *Tarjeta de Ayuda* como BusqEnTEMARIO*HelpEvent*.
 - **Botones normales:** Simplemente contienen una “cadena de texto” que será utilizada para desencadenar información.

La información al pulsar un botón se almacena en la actividad de entrada, concretamente en *Activity.Value*.

Los botones seguirán siempre presentes en la conversación. En caso de no utilizarlos no ocurrirá nada.



Botones de elección

Se insertan junto a una pregunta como mensaje al Usuario con opciones a elegir. Al elegir una de las opciones, se enviará como un mensaje al Bot y este desencadenará la acción pertinente.

La opción seleccionada se almacena en la actividad de entrada, concretamente en *Activity.Text*.

Los botones de elección desaparecen una vez se selecciona uno o se introduce un texto escrito. Si se opta por escribir, se deberá escribir el mismo texto de uno de los botones. Si no se da una contestación válida, no se podrá continuar con la conversación y se lanzará de nuevo la pregunta con los botones de elección.

Tarjetas adaptativas

El Bot se constituye casi en su totalidad de *tarjetas adaptativas*. La mejor forma de representar la información sobre el APRENDIZAJE DE C sin duda es mediante este tipo de tarjetas enriquecidas, no existe otra opción que encapsule texto (con marcaje), con imágenes, enlaces externos y bloques para sugerencias de forma estructurada.

Para cada punto del APRENDIZAJE DE C se utiliza una *tarjeta adaptativa*, que marca su estilo de presentación. Evidentemente no se va a describir el estilo seguido por cada una, pero, a grosso modo, se puede decir que cada tarjeta del APRENDIZAJE DE C, tanto para el TEMARIO como para los PROBLEMAS, sigue un estilo parecido. El resto de las tarjetas como las *Tarjetas principales* siguen estilos diferentes.

Tarjetas principales

Al grupo de tarjetas con más relevancia se les denomina *Tarjetas principales* y está compuesto de las tarjetas: *índice del TEMARIO*, *índice de los PROBLEMAS*, *Ayuda*, *Bibliografía* y *agradecimientos*, además de la ya descrita *Tarjeta de Introducción*.

Índice del TEMARIO



TEMARIO

PARTE I. METODOLOGÍA DE LA PROGRAMACIÓN

Capítulo 1 **Introducción a la ciencia de la computación y a la programación**

Introducción

PARTE I. METODOLOGÍA DE LA PROGRAMACIÓN

Capítulo 2 **Fundamentos de la programación**

Introducción

Programación estructurada

Diagrama de flujo



Ilustración 8.- Carrusel del Índice del TEMARIO

Ilustración 8.- Carrusel del Índice del TEMARIO:

Muestra el índice de forma concisa con un apartado visual minimalista y claro. Sigue un formato de tipo carrusel de tarjetas con recorrido a derechas. Cada tarjeta/capítulo contiene la parte, el capítulo, los apartados o puntos, los subapartados y los ejercicios relacionados si corresponde. Se puede desatar mediante el botón correspondiente de la *Tarjeta de Introducción* o escribiendo “índice del temario”.



Índice de los PROBLEMAS

PROBLEMAS

PARTE II. FUNDAMENTOS DE PROGRAMACIÓN EN C

Capítulo 3 Elementos básicos del lenguaje C

Capítulo 4 Operadores y expresiones

Problema 1

Problema 2

Problema 3

Problema 4

Problema 5

Problema 6

Problema 7

Problema 8

Problema 9

Problema 10

Problema 11

Problema 12

Problema 13

PARTE II. FUNDAMENTOS DE PROGRAMACIÓN EN C

Capítulo 5 Estructuras de selección: sentencias if y switch

Problema 14

Problema 15

Problema 16

Problema 17

Problema 18

Problema 19

Problema 20

Problema 21

Problema 22

Problema 23

Problema 24

Problema 25

Problema 26

Problema 27

Ilustración 9.- Carrusel del Índice de los PROBLEMAS

Ilustración 9.- Carrusel del Índice de los PROBLEMAS:

Sigue el mismo estilo que el índice del TEMARIO en minimalismo, claridad, formato tipo carrusel, ... salvo en el contenido de cada tarjeta, que contendrá la parte, el capítulo y los problemas por capítulo. Se puede desatar desde la *Tarjeta de Introducción* o escribiendo "índice de los problemas".



Ayuda

A	¿En qué puedo ayudarte?
Y	<i>Info. Temario.</i> Objetivos, características y
U	organización del contenido
D	<i>Info. Problemas.</i> Características y organización del
A	contenido
<i>Pregunta.</i> Cómo buscar apartados del Temario	
<i>Pregunta.</i> Cómo buscar problemas	
<i>Pregunta.</i> Diferenciación entre ejercicios y	
problemas	
<i>Pregunta.</i> ¿Se almacena el último punto consultado	
una vez cerrada la aplicación?	

Ilustración 10.- Tarjeta de Ayuda

Ilustración 10.- Tarjeta de Ayuda:

La *Tarjeta de Ayuda* muestra la ayuda de forma clara con un estilo más marcado (de un vistazo se visualiza la tarjeta que es). Cada punto de la información proporcionada posee un texto de fácil comprensión y está distribuido en formato de filas. Se puede desatar desde la *Tarjeta de Introducción* o escribiendo “ayuda”.

Información detallada

Muestra la tarjeta de ayuda con las diferentes opciones a elegir. Dichas opciones son seleccionables. Las opciones contenidas son:

- **Info. TEMARIO. Objetivos, características y organización del contenido.** Muestra toda la información relacionada con el TEMARIO como la orientación y objetivos, características y organización del contenido.
- **Info. PROBLEMAS. Características y organización del contenido.** Muestra toda la información relacionada con los PROBLEMAS como las características y la organización del contenido.
- **Pregunta. Cómo buscar apartados del TEMARIO.** Muestra información detallada sobre como buscar un apartado del TEMARIO además de mostrar algunos ejemplos de funcionamiento.
- **Pregunta. Como buscar problemas.** Muestra información detallada sobre como buscar un problema de los PROBLEMAS además de mostrar algunos ejemplos de funcionamiento.

- **Pregunta. Diferenciación entre ejercicios y problemas.** Describe de manera precisa la diferencia que existe entre los Ejercicios incluidos en el TEMARIO y problemas incluidos en los PROBLEMAS.
- **Pregunta. ¿Se almacena el último punto consultado una vez cerrada la aplicación?** Da respuesta a la pregunta propuesta.

Bibliografía y agradecimientos

B Agradecimientos

I Al igual que cualquier otra obra, un **Bot** como *aplicación informática* no está creado por los **B** conocimientos de una sola persona, sino que se nutre de muchas otras partes como - otras personas, **L** información de sitios web, libros, y demás -. Sin más **I** demora agradecer a todos esas partes que hicieron posible este proyecto:

O

G Realizador

R Isidro Gómez Plasencia

A Profesor director del proyecto

F Diego Alonso Cáceres

I Libros y sitios Web

A Temario de C

Libro

PROGRAMACIÓN EN C. Luis Joyanes Aguilar, Ignacio Zahonero Martínez

tutorialspoint

Sitio Web:
<https://www.tutorialspoint.com/cprogramming>

Ilustración 11.- Parte de la Bibliografía y agradecimientos

Ilustración 11.- Parte de la Bibliografía y agradecimientos:

Sigue un estilo parecido al de la *Tarjeta de Ayuda*, pero mostrando el contenido directamente. En primer lugar, hace un comentario general de agradecimiento a todas las partes implicadas en el proyecto, en segundo lugar, muestra el realizador y profesor director del proyecto, por último, describe la bibliografía (libros y sitio Web) seguida en cada parte del proyecto (TEMARIO y PROBLEMAS de C, Aprendizaje de C# y Aprendizaje sobre Bots). Se puede desatar desde la *Tarjeta de Introducción* o escribiendo “bibliografia”.

Almacenamiento

De la forma en la que se ha encaminado el Bot, no necesita guardar ningún tipo de estado y por lo tanto no dispone de almacenamiento persistente. La implementación sencilla consigue un acceso rápido a cualquier punto del APRENDIZAJE DE C y permite la continuación al siguiente de forma recurrente.

Implementa una variable que almacena el “último apartado consultado”. Se puede pensar que esta variable está más orientada al almacenamiento persistente, pero tiene el objetivo de poder consultar la Ayuda, y otra información externa, para luego volver a la lección por donde se dejó.

El Bot utiliza una función de autoguardado en el *middleware* (*AutoSaveStateMiddleware*) para almacenar los estados del Usuario y de la Conversación en cada turno de la conversación, pero ¡jojo! solo mientras la aplicación siga abierta.

En próximas actualizaciones sería interesante implementar un almacenamiento persistente y registrar a los usuarios para guardar sus estados asociados, implementar un historial, etc.

Pila de diálogos

Partiendo de la codificación más sencilla posible unida a la buena experiencia de usuario, se llegó a la conclusión de que, con **dos** diálogos en la pila (tres cuando se quiere cancelar el segundo), sería suficiente para cumplir con la exposición de información sobre el APRENDIZAJE DE C. Un diálogo para el *Main* y un segundo diálogo para exponer un apartado del APRENDIZAJE DE C.

El primer diálogo sirve de “columna vertebral” de distribución y búsqueda de información y el segundo diálogo será una de las “extremidades” con la información perteneciente a un punto del APRENDIZAJE DE C.

Debido a la cantidad de información que se expone al usuario y el requisito de dividirla en partes (modularla), hace que el *diálogo de componentes* sea el tipo de diálogo más eficiente.

Importante: absolutamente todos los diálogos de los que se compone el Bot derivan, de algún modo, del diálogo de componentes. El Main es, a nivel de código, el diálogo base del cual emanan todos los demás, en su gran mayoría los del APRENDIZAJE DE C.

Inserción del primer diálogo

Al recibir una actualización de la conversación (*ConversationUpdate*) o un mensaje de entrada del Usuario, se inserta automáticamente el primer diálogo en la pila, el diálogo *Main*, considerado el diálogo base. Para:

- La actualización de la conversación – *ConversationUpdate*: Se lanzará la *Tarjeta de Introducción* que servirá de guía al Usuario en el manejo del Bot.
- El mensaje de entrada, desatado mediante la pulsación de un botón o escribiendo, respectivamente lanzará un evento especial o se estudiará la información recogida en el mensaje (por ejemplo, mediante *Luis*) para después actuar en consecuencia.

Inserción del segundo diálogo

Si se ha pulsado un botón o se ha escrito texto, se creará el mensaje de entrada. El mensaje será analizado, y si es correcto, se insertará un segundo diálogo en la pila a la vez que se mostrará la información pertinente.

Una vez expuesta la información se propondrá:

- Reemplazar el diálogo actual (segundo en la pila) por otro: continuar al siguiente punto del APRENDIZAJE DE C.
- Cancelar el diálogo actual (segundo en la pila): Sacar el diálogo actual de la pila volviendo al primero (*Main*) y permitiendo buscar otra información.
- Otras opciones: que dependen del punto del APRENDIZAJE DE C consultado como mostrar el índice al siguiente capítulo, mostrar la solución o explicación de un ejercicio o problema, etc.

El segundo diálogo estará relacionado principalmente con el APRENDIZAJE DE C, es decir, con un punto del TEMARIO o de los PROBLEMAS.

Comprendión del lenguaje

Para hacer que el Bot sea más inteligente o lo que es lo mismo mejor conversador, la herramienta más eficaz es el uso de servicios de comprensión del lenguaje.

El servicio de comprensión del lenguaje que se ha usado es **LUIS** (language understanding). Dicho servicio cumple con los requisitos de diseño perfectamente, además es un servicio gratuito (hasta un límite de 1000 utilizaciones o reconocimientos de texto).

Un procesador del lenguaje se encarga de determinar una intención (*intent*) a partir de una expresión de entrada (*utterance*). Cada intención recoge las diferentes expresiones asociadas (por ejemplo, la intención “Problema9” recoge las expresiones: “problema 9” y “problema nueve”) y si una de esas expresiones coincide con el mensaje de entrada, el servicio devuelve la intención (“Problema9”) al código del Bot (mediante esa intención el Bot actuará en consecuencia).

El Bot utiliza 4 aplicaciones o modelos LUIS: un distribuidor a otros modelos, uno de uso general que contiene las opciones más comunes y otros 2 que contienen toda la información sobre el APRENDIZAJE DE C.

El modelo **Dispatch**, denominado *modelo distribuidor*, es un encaminador de modelos que contiene toda la información de cada modelo agrupada en intenciones (en concreto 3 intenciones correspondientes a los 3 *modelos principales*). Al recibir un mensaje de entrada, **Dispatch**, emite una intención que corresponde al modelo donde se encuentra dicha expresión, por ejemplo, para el mensaje de entrada “capítulo 4 operadores lógicos”, **Dispatch**, devuelve una intención “**TemarioC**” porque dicha expresión corresponde a ese modelo.

Los otros 3 *modelos principales* son respectivamente:

- **General:** permite desencadenar la funcionalidad general de Bot como crear un nuevo usuario, acceder a la Tarjeta de Ayuda y a la Tarjeta de Introducción, mostrar el último punto consultado, mostrar la Bibliografía, etc.

- **TemarioC:** permite desencadenar toda la información respectiva al TEMARIO del APRENDIZAJE DE C. Información ordenada en capítulos, apartados de capítulo, subapartados de apartados y ejercicios de fin de capítulo.
- **ProblemasC:** permite desencadenar todos los PROBLEMAS del APRENDIZAJE DE C. Información ordenada en capítulos y problemas por capítulo.

Interrupciones

El control de interrupciones se realiza solo cuando se ejecuta un *diálogo en cascada* como algún punto del TEMARIO o de los PROBLEMAS. En las demás partes... que principalmente es el Main, no es necesario controlar las interrupciones porque su objetivo es el de buscar información no el de procesarla.

El Bot contiene dos interrupciones: **cancelar** y **ayuda**.

- La interrupción de cancelación se encarga de cancelar el diálogo activo, es decir, el segundo diálogo en la pila de diálogos. Tras la cancelación se permite buscar otra información.
- La interrupción de ayuda permite, primero, cancelar el diálogo activo y, después, mostrar la *Tarjeta de Ayuda*.

Resumiendo, las dos interrupciones disponibles cancelan, de un modo u otro, el diálogo activo.

En caso de querer consultar la ayuda y posteriormente continuar la lección, primero, habrá que escribir “ayuda”, segundo, cancelar el diálogo activo mostrando así la *Tarjeta de Ayuda*, y, tercero, decirle al Bot que muestre el último apartado consultado escribiendo por ejemplo “mostrar anterior” o “continuar”.

APRENDIZAJE DE C. Tipos de contenido, tablas de contenido, archivado y procesado

El contenido sobre el APRENDIZAJE DE C está bien ordenado y estructurado en la carpeta que contiene los diálogos del Bot (*Dialogs*) repartido entre la carpeta “TEMARIO” y la carpeta “PROBLEMAS”.

A continuación, se explica todo lo referente a los tipos de contenido, las tablas de contenido, el archivado y el procesamiento que sigue todo el APRENDIZAJE DE C.

Tipos de contenido

El TEMARIO explica cada apartado y subapartado mediante definiciones que incluyen titulación, texto con marcaje, imágenes, notas y/o avisos. Además, incluye Ejercicios de fin de capítulo que ayudan a comprender lo explicado en cada capítulo.

- **Explicación de apartado.** Como se puede ver en la Ilustración 12.- Explicación de apartado, la explicación de cada apartado o subapartado de un capítulo puede contar con titulación, texto, imágenes, énfasis, viñetas, etc. Se da opción de continuar o no con el siguiente.

Welcome Live Chat (developm... x)

[Restart conversation](#) [Save transcript](#)

http://localhost:3978/api/messages

Capítulo 3. Elementos básicos del lenguaje C

Directivas del preprocesador

Las directivas del preprocesador incluyen `#include` y `#define`, además NO terminan en `()` como les ocurre a las sentencias. El símbolo almohadilla (`#`) indica al compilador que lea e incluya el contenido de esas directivas en dicha posición. Esta sintaxis, distinta a la del lenguaje C, se debe a que el preprocesador no es estrictamente parte del conjunto de herramientas del lenguaje.

Su uso más común son los *archivos de cabecera* como por ejemplo `stdio.h`, `stdlib.h`, `math.h` y `string.h` incluidos entre caracteres mayor-menor `<...>`, además de los *archivos cabecera creados por el usuario* que irán entre comillas `"..."` y serán buscados en la carpeta en la que se encuentra el programa creado.

```
#include <archivo_estandar_de_C.h>
#include "archivo_creado_por_el_usuario.h"
```

Nota: Cuando se instala el compilador de C, automáticamente se instalan

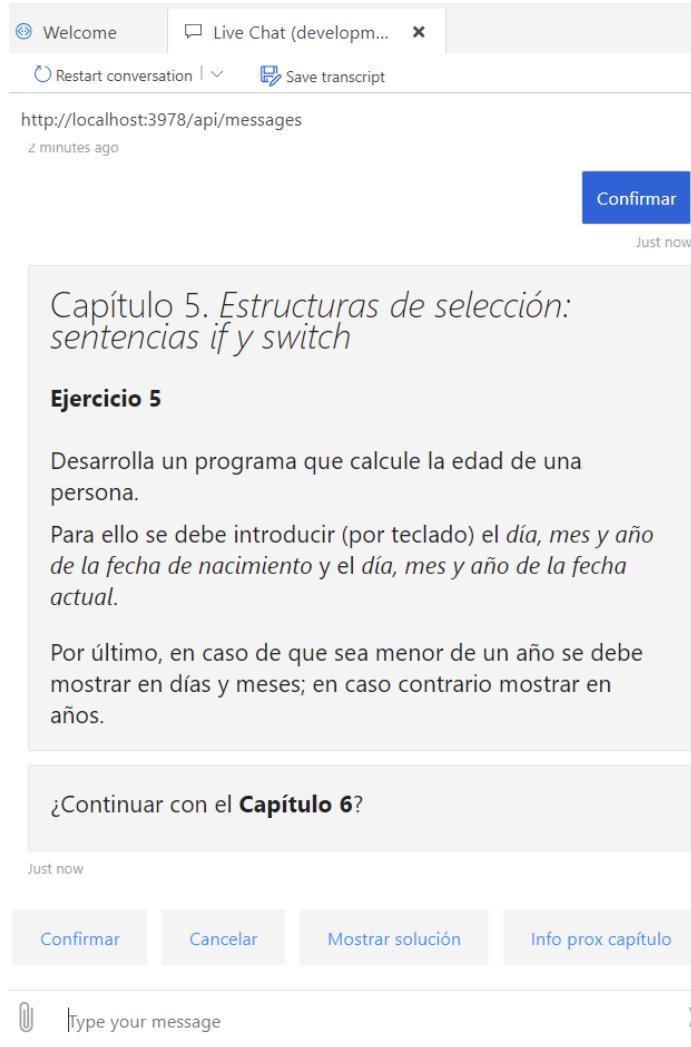
New messages

Confirmar Cancelar

Type your message ➤

Ilustración 12.- Explicación de apartado

- **Enunciado y solución de Ejercicio.** La Ilustración 13.- Enunciado y solución de Ejercicio muestra el enunciado del ejercicio correspondiente permitiendo al Usuario realizarlo por su cuenta. Se da la opción de mostrar la solución, pasar al siguiente apartado y en algunos casos permite mostrar el índice al siguiente capítulo.



The screenshot shows a live chat window with the following details:

- Header:** Welcome, Live Chat (developm...), Close, Restart conversation, Save transcript.
- Message:** http://localhost:3978/api/messages, 2 minutes ago.
- Bot Response:** Just now, **Confirmar**.
- Content Area:**

**Capítulo 5. Estructuras de selección:
sentencias if y switch**

Ejercicio 5

Desarrolla un programa que calcule la edad de una persona.

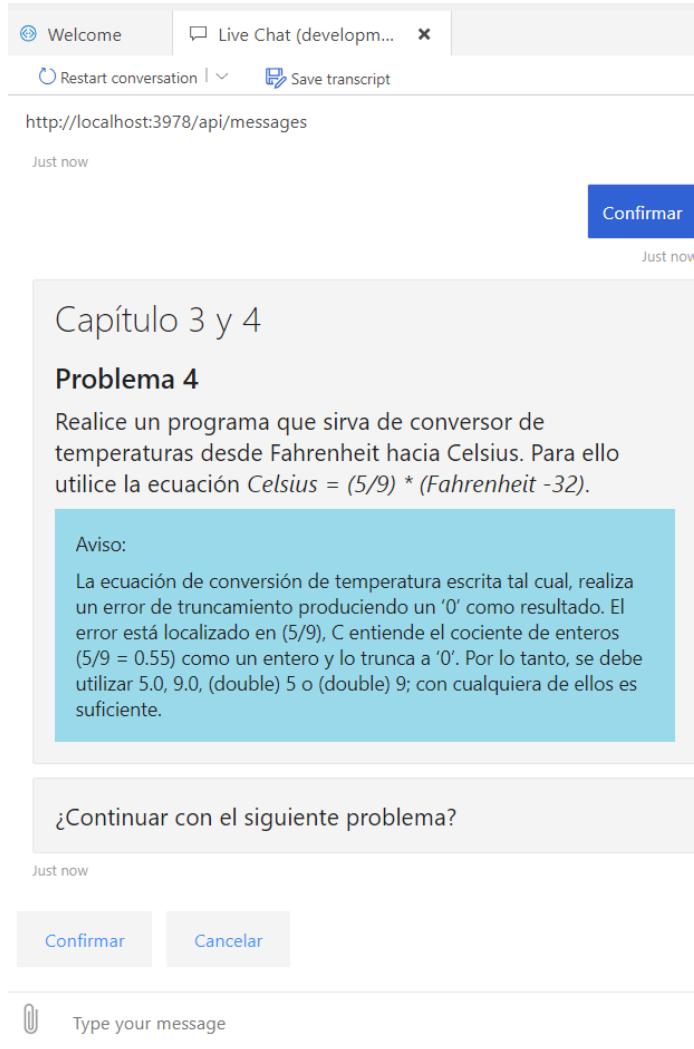
Para ello se debe introducir (por teclado) el *día, mes y año de la fecha de nacimiento* y el *día, mes y año de la fecha actual*.

Por último, en caso de que sea menor de un año se debe mostrar en días y meses; en caso contrario mostrar en años.
- Footer:** Just now, **¿Continuar con el Capítulo 6?**, **Confirmar**, **Cancelar**, **Mostrar solución**, **Info prox capítulo**.
- Input Field:** Type your message.

Ilustración 13.- Enunciado y solución de Ejercicio

La implementación de los PROBLEMAS requería de seguir unos tipos bien marcados, es decir, si se aceptase cualquier tipo de problema, se originaría una implementación muy tediosa y lenta. Así, se crearon tres tipos de problemas bien diferenciados que servían como plantilla para los demás. Entre los tres tipos se encuentran problemas para realizar, explicados y de solución desbloqueable.

- **Problemas para realizar.** La Ilustración 14.- Problema para realizar muestra el enunciado del problema y todos los requisitos a seguir para su realización. Se da la opción de pasar o no al siguiente problema.

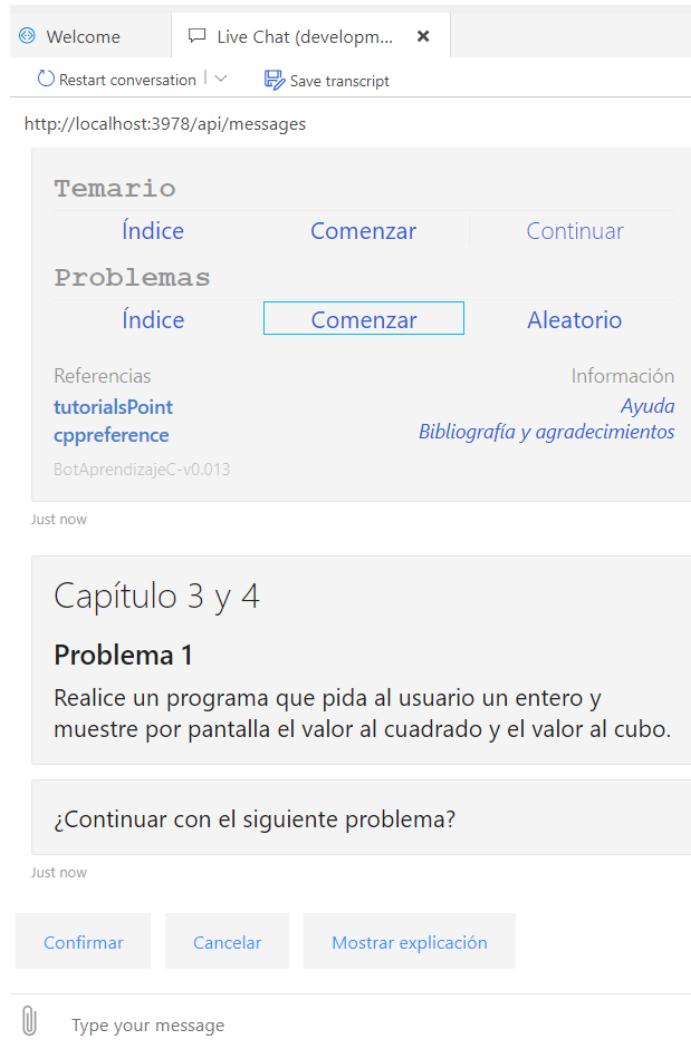


The screenshot shows a live chat interface with the following elements:

- Header:** Welcome, Live Chat (developm...), Restart conversation, Save transcript.
- URL:** http://localhost:3978/api/messages
- Text:** Just now
- Buttons:** Confirmar (blue button), Just now
- Section:** Capítulo 3 y 4
- Section:** Problema 4
- Text:** Realice un programa que sirva de conversor de temperaturas desde Fahrenheit hacia Celsius. Para ello utilice la ecuación $Celsius = (5/9) * (Fahrenheit - 32)$.
- Text:** Aviso: La ecuación de conversión de temperatura escrita tal cual, realiza un error de truncamiento produciendo un '0' como resultado. El error está localizado en (5/9), C entiende el cociente de enteros (5/9 = 0.55) como un entero y lo trunca a '0'. Por lo tanto, se debe utilizar 5.0, 9.0, (double) 5 o (double) 9; con cualquiera de ellos es suficiente.
- Text:** ¿Continuar con el siguiente problema?
- Buttons:** Confirmar, Cancelar
- Input Field:** Type your message

Ilustración 14.- Problema para realizar

- **Problemas explicados.** La Ilustración 15.- Problema explicado muestra el enunciado del problema dando la opción de realizarlo. A parte, se la opción de mostrar la explicación o pasar al siguiente problema.



Welcome Live Chat (developm... x)

Restart conversation | Save transcript

http://localhost:3978/api/messages

Temario

Índice Comenzar Continuar

Problemas

Índice Comenzar Aleatorio

Referencias Información

tutorialsPoint Ayuda

cppreference Bibliografía y agradecimientos

BotAprendizajeC-v0.013

Just now

Capítulo 3 y 4

Problema 1

Realice un programa que pida al usuario un entero y muestre por pantalla el valor al cuadrado y el valor al cubo.

¿Continuar con el siguiente problema?

Just now

Confirmar Cancelar Mostrar explicación

Type your message ➤

Ilustración 15.- Problema explicado

- **Problemas de solución desbloqueable.** La Ilustración 16.- Problemas de solución desbloqueable muestra el enunciado del problema además de una o varias cuestiones a responder. En función de los aciertos se mostrará un tipo de respuesta. Una vez se ha contestado, se da la opción de pasar al siguiente problema.

Welcome **Live Chat (developm... x**

Restart conversation | **Save transcript**

http://localhost:3978/api/messages

Just now

BotAprendizajeC-v0.013

Just now

problema 10

Just now

Capítulo 3 y 4

Problema 10

Según el código que se muestra abajo...

```
1 #include <stdio.h>
2 #include <stdint.h>
3 void main()
4 {
5     int16_t a = 0xFFFF4, b = 12, c = a == -b ? 1 : 0;
6     printf("%hd\n%hd\n%hd", a, b, c);
7 }
```

¿Qué valor se mostrará al imprimir *a*?

-12

31

23

Just now

Type your message

Welcome **Live Chat (developm... x**

Restart conversation | **Save transcript**

http://localhost:3978/api/messages

A minute ago

¿Qué valor se mostrará al imprimir *a*?

-12

31

23

Just now

Capítulo 3 y 4

Problema 10

¡Eso es! Al realizar, compilar y ejecutar el código se comprueba que *a* vale -12. Si el tipo de la variable fuese *int* (equivalente a *int32_t*) el resultado sería el mismo, simplemente se muestra el tipo *int16_t* para hacer entender que existen otros tipos de datos.

¿Continuar con el siguiente problema?

Just now

Confirmar **Cancelar**

Type your message

Ilustración 16.- Problemas de solución desbloqueable

Tablas de contenido

Las tablas de contenido muestran un resumen del contenido del APRENDIZAJE DE C como los pasos de cascada y los botones de elección de cada apartado, si se almacena el último diálogo consultado, si se muestra enunciado en los Ejercicios y los tipos de PROBLEMAS.

APARTADOS DEL TEMARIO PARA EL:	Pasos de cascada	Elecciones	Almacena último diálogo consultado
Capítulo 1	3	Confirmar, Cancelar, Índice	Si
Capítulo 2	2	Confirmar, Cancelar	Si
	(3) ¹	(Confirmar, Cancelar, Índice) ¹	Si
Capítulo 3	2	Confirmar, Cancelar	Si
Capítulo 4	2	Confirmar, Cancelar	Si
Capítulo 5	2	Confirmar, Cancelar	Si
Capítulo 6	2	Confirmar, Cancelar	Si
Capítulo 7	2	Confirmar, Cancelar	Si
Capítulo 8	2	Confirmar, Cancelar	Si
Capítulo 9	2	Confirmar, Cancelar	Si

¹: Último apartado del capítulo.

EJERCICIOS DEL TEMARIO	Pasos de cascada	Elecciones		Muestra Enunciado	Almacena último diálogo consultado
Capítulo 3					
Ejercicio 1	3	Confirmar, Cancelar, Índice		No	Si
Capítulo 4					
Ejercicio 2	3	Confirmar, Cancelar, Solución		Si	Si
Ejercicio 3	4	Confirmar, Cancelar, Solución, Índice		Si	Si
Capítulo 5					
Ejercicio 4	3	Confirmar, Cancelar, Solución		Si	Si
Ejercicio 5	4	Confirmar, Cancelar, Solución, Índice		Si	Si
Capítulo 6					
Ejercicio 6	3	Confirmar, Cancelar, Solución		Si	Si
Ejercicio 7	3	Confirmar, Cancelar, Solución		Si	Si
Ejercicio 8	3	Confirmar, Cancelar, Solución		Si	Si
Ejercicio 9	4	Confirmar, Cancelar, Solución, Índice		Si	Si
Capítulo 7					
Ejercicio 10	2	Confirmar, Cancelar		No	Si
Ejercicio 11	3	Confirmar, Cancelar, Solución		Si	Si
Ejercicio 12	4	Confirmar, Cancelar, Solución, Índice		Si	Si
Capítulo 8					
Ejercicio 13	3	Confirmar, Cancelar, Solución		Si	Si
Ejercicio 14	3	Confirmar, Cancelar, Solución		Si	Si
Ejercicio 15	3	Confirmar, Cancelar, Solución		Si	Si
Ejercicio 16	3	Confirmar, Cancelar, Solución		Si	Si
Ejercicio 17	4	Confirmar, Cancelar, Solución, Índice		Si	Si

PROBLEMAS	Pasos de cascada	Elecciones		Tipo	Almacena último diálogo consultado
Capítulo 3 y 4					
Problema 1	3	Confirmar, Cancelar, Explicación		Explicar	Si
Problema 2	3	Opciones	Confirmar, Cancelar	Desbloquear	Si
Problema 3	3	Confirmar, Cancelar, Explicación		Explicar	Si
Problema 4	2	Confirmar, Cancelar		Realizar	Si
Problema 5	3	Opciones	Confirmar, Cancelar	Desbloquear	Si
Problema 6	3	Confirmar, Cancelar, Explicación		Explicar	Si

Problema 7	2	Confirmar, Cancelar		Realizar	Si
Problema 8	3	Confirmar, Cancelar, Explicación		Explicar	Si
Problema 9	2	Confirmar, Cancelar		Realizar	Si
Problema 10	3	Opciones	Confirmar, Cancelar	Desbloquear	Si
Problema 11	3	Opciones	Confirmar, Cancelar	Desbloquear	Si
Problema 12	3	Confirmar, Cancelar, Explicación		Explicar	Si
Problema 13	2	Confirmar, Cancelar		Realizar	Si
Capítulo 5					
Problema 14	2	Confirmar, Cancelar		Realizar	Si
Problema 15	2	Confirmar, Cancelar		Realizar	Si
Problema 16	3	Confirmar, Cancelar, Explicación		Explicar	Si
Problema 17	3	Opciones	Confirmar, Cancelar	Desbloquear	Si
Problema 18	3	Confirmar, Cancelar, Explicación		Explicar	Si
Problema 19	3	Confirmar, Cancelar, Explicación		Explicar	Si
Problema 20	2	Confirmar, Cancelar		Realizar	Si
Problema 21	2	Confirmar, Cancelar		Realizar	Si
Problema 22	2	Confirmar, Cancelar		Realizar	Si
Problema 23	3	Confirmar, Cancelar, Explicación		Explicar	Si
Problema 24	3	Opciones	Confirmar, Cancelar	Desbloquear	Si
Problema 25	3	Confirmar, Cancelar, Explicación		Explicar	Si
Problema 26	2	Confirmar, Cancelar		Realizar	Si
Problema 27	3	Confirmar, Cancelar, Explicación		Explicar	Si
Problema 28	3	Opciones	Confirmar, Cancelar	Desbloquear	Si
Problema 29	3	Confirmar, Cancelar, Explicación		Explicar	Si
Capítulo 6					
Problema 30	3	Opciones	Confirmar, Cancelar	Desbloquear	Si
Problema 31	2	Confirmar, Cancelar		Realizar	Si
Problema 32	2	Confirmar, Cancelar		Realizar	Si
Problema 33	3	Opciones	Confirmar, Cancelar	Desbloquear	Si
Problema 34	2	Confirmar, Cancelar		Realizar	Si
Problema 35	2	Confirmar, Cancelar		Realizar	Si
Problema 36	3	Confirmar, Cancelar, Explicación		Explicar	Si
Problema 37	2	Confirmar, Cancelar		Realizar	Si
Problema 38	3	Confirmar, Cancelar, Explicación		Explicar	Si
Problema 39	2	Confirmar, Cancelar		Realizar	Si
Problema 40	3	Confirmar, Cancelar, Explicación		Explicar	Si
Problema 41	3	Opciones	Confirmar, Cancelar	Desbloquear	Si
Problema 42	2	Confirmar, Cancelar		Realizar	Si
Problema 43	2	Confirmar, Cancelar		Realizar	Si
Problema 44	3	Opciones	Confirmar, Cancelar	Desbloquear	Si
Capítulo 7					
Problema 45	3	Opciones	Confirmar, Cancelar	Desbloquear	Si
Problema 46	2	Confirmar, Cancelar		Realizar	Si
Problema 47	3	Confirmar, Cancelar, Explicación		Explicar	Si
Problema 48	3	Confirmar, Cancelar, Explicación		Explicar	Si
Problema 49	2	Confirmar, Cancelar		Realizar	Si
Problema 50	3	Confirmar, Cancelar, Explicación		Explicar	Si

<i>Problema 51</i>	3	Confirmar, Cancelar, Explicación		Explicar	Si
Capítulo 8					
<i>Problema 52</i>	3	Confirmar, Cancelar, Explicación		Explicar	Si
<i>Problema 53</i>	2	Confirmar, Cancelar		Realizar	Si
<i>Problema 54</i>	2	Confirmar, Cancelar		Realizar	Si
<i>Problema 55</i>	3	Confirmar, Cancelar, Explicación		Explicar	Si
<i>Problema 56</i>	2	Confirmar, Cancelar		Realizar	Si
<i>Problema 57</i>	2	Confirmar, Cancelar		Realizar	Si
<i>Problema 58</i>	3	Opciones	Confirmar, Cancelar	Desbloquear	Si
<i>Problema 59</i>	3	Opciones	Confirmar, Cancelar	Desbloquear	Si
<i>Problema 60</i>	2	Confirmar, Cancelar		Realizar	Si
<i>Problema 61</i>	3	Opciones	Confirmar, Cancelar	Desbloquear	Si
<i>Problema 62</i>	2	Confirmar, Cancelar		Realizar	Si
<i>Problema 63</i>	2	Confirmar, Cancelar		Realizar	Si
<i>Problema 64</i>	3	Opciones	Confirmar, Cancelar	Desbloquear	Si
<i>Problema 65</i>	3	Confirmar, Cancelar, Explicación		Explicar	Si
Capítulo 9					
<i>Problema 66</i>	3	Confirmar, Cancelar, Explicación		Explicar	Si
<i>Problema 67</i>	2	Confirmar, Cancelar		Realizar	Si
<i>Problema 68</i>	3	Confirmar, Cancelar, Explicación		Explicar	Si
<i>Problema 69</i>	3	Opciones	Confirmar, Cancelar	Desbloquear	Si

Archivado

Cada punto del TEMARIO o de los PROBLEMA contiene una serie de archivos y carpetas, bien definidos, entre los que se encuentran *archivos de recursos (.resx)*, *archivos de notación (.json)* y *archivos de código (.cs)*.

- **Archivos de código (.cs):** son principalmente dos:
 - ...*Dialog.cs*: contiene la estructura que sigue cada apartado como la lógica, diálogos en cascada, diálogos de aviso, el almacenamiento del último apartado consultado y las diferentes llamadas al archivo de respuestas (...Responses.cs) o a las posibles respuestas (.resx).
 - ...*Responses.cs*: clase derivada del administrador de plantillas (*TemplateManger*) que gestiona las respuestas de cada parte del Bot (*Main*, puntos del APRENDIZAJE DE C, *middleware*, etc).
- **Archivos de recursos (.resx):** usados para almacenar rutas de enlace a otros archivos (como .json), así como texto (por ejemplo, texto normal y texto con marcaje MARKDOWN).
- **Archivos de notación (.json):** estos archivos contienen la información de cada punto del APRENDIZAJE DE C convertida a código .json; admiten notación HTML y marcaje MARKDOWN. Particularmente utilizan notación .json para tarjetas adaptativas.

Procesado

Una vez se solicita al Bot que muestre un punto del APRENDIZAJE DE C, este se moverá por la lógica del punto (...*Dialog.cs*) desencadenando los pasos de la cascada. Existen dos tipos de paso de cascada diferenciados:



- **Paso – muestra de información:** como se muestra en Ilustración 17.- Muestra de información, contendrá principalmente una respuesta con el contenido del punto (contenido de un punto del APRENDIZAJE DE C) y una respuesta junto a una elección (pregunta con opciones a elegir, del estilo, ¿Continuar al siguiente punto? Si, No).

Capítulo 2. Fundamentos de programación

Programación estructurada

Es una combinación de programación modular y *estructuras de control*. Hace que los programas sean más fáciles de escribir, verificar, leer y mantener. Una estructura de control es una estructura marcada con una función específica que permite modificar el flujo de ejecución de las instrucciones. Los tres tipos básicos son secuencia, selección y repetición.

¿Continuar con el siguiente apartado?

2 hours ago

[Confirmar](#)
[Cancelar](#)

Ilustración 17.- Muestra de información

- **Paso – finalización de diálogo:** mediante la respuesta al paso anterior se podrá continuar al siguiente punto del APRENDIZAJE DE C, reemplazando el diálogo actual por el siguiente. O, en el control de interrupciones, se podrá cancelar el diálogo activo.

Nota: Ambos tipos de pasos de cascada puede ser utilizados varias veces para conseguir funcionalidades adaptadas a cualquier tipo de situación.

Adicionalmente, el primer paso de la cascada almacenará el último apartado consultado.

Por último, dejar constancia de que, existen dos carpetas que contienen la información compartida para cada punto, una para el TEMARIO y otra para los PROBLEMAS.

Software intermedio o middleware

Hay aspectos que no pueden ser tratados o que no cumplen totalmente con el cometido deseado, dentro de la lógica del Bot. Por suerte existe una parte de la arquitectura Bot que cumple con estos requerimientos y es el *software intermedio o middleware*. Este software se sitúa entre el Bot y el Usuario, y actúa cuando se envían mensajes de un extremo al otro, es decir, cuando el Usuario envía un mensaje al Bot o el Bot envía un mensaje al Usuario. Se podría decir que es como una clase que se ejecuta cuando se envía un mensaje y puede actuar sobre él o partir de él.

Las herramientas que contiene el *middleware* se ejecutan en serie y deben ser dispuestas siguiendo un orden de prioridad claro, por relevancia o dependencia. Las funciones o herramientas ubicadas en el *middleware*, para el Bot que se ha querido crear, son:

- **OnTurnError.** Captura cualquier error de funcionamiento que ocurra en cualquier turno de la conversación. Además, muestra una aproximación del error (mensaje y fuente) dando a entender cuál puede haber sido el fallo. Esta herramienta está más orientada al desarrollador que al usuario.
- **ShowTypingMiddleware.** Muestra visualmente cuando el Bot está trabajando o respondiendo a una pregunta (mediante un ícono en movimiento). Esta herramienta se hace visible en el turno de conversación del Bot.
- **SetLocaleMiddleware.** Fija la localización actual mediante conexión a los servicios de Azure de Microsoft o mediante la ubicación por defecto (es-ES). La localización determina el idioma utilizado por el Bot (mensajes de salida, gestores de respuestas, *archivos de recursos*, *archivos de notación*, comprensión del lenguaje, ...) y por Usuario. Por ahora solo esta soportado el idioma español, aunque la metodología de construcción soporta la implantación de otros idiomas.
- **AutoSaveStateMiddleware.** Fija y almacena el estado de la Conversación y el estado del Usuario en cada turno de conversación. Se hace uso del autoguardado porque la metodología de construcción del Bot no necesita de un guardado manual.

2.2.3. Herramientas de construcción

Más adelante, se describen todas las herramientas o aplicaciones informáticas que de alguna manera se han usado para la realización del proyecto. Entre ellas se encuentran aplicaciones Web, procesadores de texto, aplicaciones PDF, almacenes en la nube de imágenes procesables y de código C, entre otras.

Herramientas de escritura de código

- Código del Bot ([C#](#)) → VISUAL STUDIO 2017 COMMUNITY.

El lenguaje de programación utilizado para la construcción del Bot es C# debido a que la documentación, que Microsoft pone a disposición del consumidor, está prácticamente en su totalidad en ese lenguaje. Así, la aplicación de escritorio utilizada, que servirá como plataforma de escritura de código C#, es Visual Studio 2017 (versión Community).

- PROBLEMAS y Ejercicios de C → Code::blocks y Visual Studio Code.

Tanto los Ejercicios como los PROBLEMAS de C han sido desarrollado, depurados y ejecutados en Code::blocks. Además, se ha utilizado Visual Studio Code para extraer imágenes planas de código que serán incluidas en ejemplos de uso determinados.

Almacén de PROBLEMAS y Ejercicios de C

- Almacén de código online – GitHub → <https://github.com>.

Algún que otro código perteneciente a los PROBLEMAS o a los Ejercicios, ha sido subido a GitHub para su consulta, principalmente aquellos que sirven de explicación al Usuario. Además de servirles de referencia, pueden ser copiados directamente dando la posibilidad de ejecutarlos y probar su funcionamiento.

Herramientas de escritura y visualización de documentos

- Procesador de Texto → Word (Office 365 ProPlus).

El procesador de texto usado ha sido Word. Ha servido para redactar todo tipo de información como la del resumen sobre el APRENDIZAJE DE C (tanto para el TEMARIO como para los PROBLEMAS), la redacción de la memoria del TFG, entre otras. A parte, ha servido de apoyo para la conversión de dicho resumen a *tarjetas adaptativas*.

- Visualizador de documentos PDF → Adobe Acrobat Reader DC.

El visualizador de documentos PDF ha sido Adobe Acrobat Reader DC. Se ha usado para visualizar documentos PDF como los libro “INFORMÁTICA APLICADA. PROGRAMACIÓN EN LENGUAJE C. Pedro María Alcover Garau, UPCT” o “PROGRAMACIÓN EN C. Luis Joyanes Aguilar, Ignacio Zahonero Martínez” referenciados en Agradecimientos.

Servicios Cognitivos

- Comprensión del lenguaje - **LUIS** → <https://www.luis.ai/>.

El servicio cognitivo utilizado ha sido **LUIS**, por cumplir con las expectativas de desarrollo y ser gratuito (hasta un límite de utilizaciones). Ha sido necesario crear cuatro aplicaciones/modelos llamados **Dispatch**, **General**, **ProblemasC** y **TemarioC** para cumplir perfectamente con el cometido del Bot.

Herramientas para convertir modelos LUIS (.json) a clases de C#

- Conversor de modelos **LUIS** – LuisGen → <https://github.com/Microsoft/botbuilder-tools/tree/master/packages/LUISGen>.

Mediante la consola de Windows y una extensión (DotNet Core SDK), se pueden convertir modelos LUIS en formato .json a clases de C# de manera sencilla. Especialmente útil cuando se baja un modelo **LUIS** de la Web (en formato .json) y se quiere convertir a clase (C#) para su uso en el código del Bot.

Herramientas de gestión de tarjetas adaptativas

- Modelaje → <https://adaptivecards.io/explorer/>.

Explica toda la información perteneciente a la creación de *tarjetas adaptativas*. Incluye la explicación de todas las partes que componen una *tarjeta adaptativa* y algunos ejemplos de uso.

- Visualizador → <https://adaptivecards.io/visualizer>.

El visualizador de *tarjetas adaptativas* permite visualizar cualquier tarjeta en diferentes canales (WebChat, Cortana, Microsoft Teams, Skype, ...) y ver si lo creado se ajusta a lo que se quiere ver. Herramienta destinada a la validación de *tarjetas adaptativas*.

- Diseñador → <https://adaptivecards.io/designer>.

El diseñador de *tarjetas adaptativas* permite diseñar cualquier tarjeta, proporcionando las herramientas para colocar -bloques de texto, imágenes, videos, botones, elección, estradas de texto, entradas de número, entradas de fecha, etc- además de estructurar la información en filas y columnas, poner fondos, hacer énfasis, etc.

Herramientas de depuración y prueba de Bots

- Emulador → **Bot Framework Emulator V4**

Muestra el comportamiento visual del Bot permitiendo hacer correcciones a nivel de interfaz y de texto. Además, junto a la herramienta de depuración de **VISUAL STUDIO 2017 COMMUNITY** permite realizar pruebas de código de manera exhaustiva. Es una herramienta imprescindible para la visualización y prueba de Bots en desarrollo.

Almacén de imágenes

- Almacén de imágenes online – imgbb → <https://ibb.co>.

imgbb es un almacén gratuito online que transforma una imagen en imagen procesada a la vez que crea una URL a la misma. Es imprescindible disponer de imágenes procesadas con acceso URL para incrustarlas en las *tarjetas adaptativas* del Bot.

Conversión de texto Word a Markdown

- Conversor de Word a MARKDOWN online → <https://word-to-markdown.herokuapp.com/>

Convierte un documento *.doc* o *.docx* a marcaje **MARKDOWN**, o lo que es lo mismo, convierte títulos, cursivas, negritas, viñetas, etc a simbología interpretable.

Conversión de texto a HTML

- Conversor de texto a **HTML** online → <https://www.textfixer.com/html/convert-text-html.php>

Convierte texto plano, carácter a carácter, a **HTML** por ejemplo letras con acento y caracteres especiales. Es posible convertir texto **MARKDOWN** a **HTML**, porque **MARKDOWN** sigue una nomenclatura de caracteres.

Código UNICODE

- Tabla estandarizada de caracteres – **UNICODE** → <https://unicode-table.com/es>

En ocasiones, es necesario echar mano de la tabla estándar de caracteres, **UNICODE**, para obtener el código **HTML** de algunos caracteres particulares que, el “conversor a **HTML**” no reconoce.

3. Desarrollo e implementación del Bot

BOTAPRENDIZAJEC****. Funcionamiento detallado

3.1. Puesta en marcha de la Aplicación Bot

3.1.1. Archivo Program.cs

Para ponerse en situación, un Bot está considerado internamente y en la puesta en marcha, una aplicación Web. Ahora, como todo programa informático de C# se desencadena por la función Main del archivo **Program.cs**.

¡Ojo! no confundir el método iniciador, Main, de cualquier programa de la familia C con la clase Main concretamente llamada *MainDialog* del Bot.

```

public static void Main(string[] args)
{
    BuildWebHost(args).Run();
}

public static IWebHost BuildWebHost(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .Build();

```

Program 1.- Main

Program 1.- Main contiene la función Main que llama al método que construye un hospedaje Web, *BuildWebHost*, y lo pone en marcha. Este método:

- Crea un hospedaje Web (*WebHost*) con valores por defecto.
- Le dice al hospedaje Web que comience la inicialización por el archivo **Startup.cs**.
- Crea una interfaz de hospedaje Web (*IWebHost*) con todos los valores anteriores.

3.1.2. Archivo Startup.cs

El archivo **Startup.cs** es llamado desde el Main (Program 1.- Main) y es el encargado de inicializar la aplicación. Este archivo posee dos partes bien diferenciadas: el conjunto -campos, propiedades y constructor-, y el conjunto -configuración de los servicios y del propio Bot-.

Campos, propiedades y constructor

Tras la definición/inicialización de los campos y propiedades de la clase **Startup.cs** (Startup 1.- Campos, constructor y propiedades) es el turno del constructor.

```

private ILoggerFactory _loggerFactory;
private bool _isProduction = false;

public Startup(IHostingEnvironment env, ILoggerFactory loggerFactory)
{
    _isProduction = env.IsProduction();
    _loggerFactory = loggerFactory;

    var configuration = new ConfigurationBuilder()
        .SetBasePath(env.ContentRootPath)
        .AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
        // .AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional: true)
        .AddEnvironmentVariables();

    Configuration = configuration.Build();
}

/// <summary>
/// Propiedad que almacena la configuracion del bot escrita en "appsettings.json"
/// ademas, de las variables de ambiente y la ruta al archivo raiz.
/// </summary>
public IConfiguration Configuration { get; }

```

Startup 1.- Campos, constructor y propiedades

Según la imagen Startup 1.- Campos, constructor y propiedades, el constructor adquiere el ambiente del hospedaje (*IHostingEnvironment*) y el almacén del registro (*ILoggerFactory*). Los valores contenidos en ambos dependen de donde se realice la carga de la aplicación, es decir, de manera Local o en la Web. Si se lanza desde la Web recibirá unos valores dependientes del canal y si se lanza de forma Local se cargarán los valores por defecto. Como se trata de desarrollar un Bot, el lanzamiento será de forma Local.

1. Como se trata de un hospedaje local, el ambiente estará en *desarrollo* (*Development*) y el almacén del registro, *_loggerFactory*, no contendrá nada. Además, el campo *_isProduction* contendrá *false* porque el Bot está en *desarrollo* no en *producción*.
2. La propiedad **Configuration** es clave en la puesta en marcha de la aplicación Bot. Almacena valores como la ruta al archivo raíz (*env.ContentRootPath*), el archivo de configuración (escrito en **appsettings.json**) y las variables de ambiente (*IHostingEnvironment*). **appsettings.json** (Código 1.- Archivo appsettings.json) contiene la ruta al archivo de configuración de los servicios utilizados por el Bot, el secreto o clave y la localización por defecto.

```
{  
    "botFilePath": ".\\BotAprendizajeC.bot",  
    "botFileSecret": "",  
    "defaultLocale": "es-ES"  
}
```

Código 1.- Archivo **appsettings.json**

Configuración del Bot y los servicios que utiliza

La clase **Startup.cs** no sigue el funcionamiento de una clase normal de C# (inicialización a secas de los campos, propiedades y constructor) sino que también carga los métodos que contiene de manera secuencial.

De esta forma el método *ConfigureServices* (Startup 2.- Configuración de servicios utilizados por el Bot – ConfigureServices Parte 1 y Startup 3.- Configuración de servicios utilizados por el Bot – ConfigureServices Parte 2) es el primer método en actuar y recibe una interfaz de colección de servicios (*IServiceCollection*) que registrará todos los servicios que utilizará el Bot. Sigue los siguientes pasos:

```

/// <summary> Metodo encargado de configurar el bot y los servicios que utiliza.
public void ConfigureServices(IServiceCollection services)
{
    var botFilePath = Configuration.GetSection("botFilePath")?.Value;
    var botFileSecret = Configuration.GetSection("botFileSecret")?.Value;

    // Carga los 'servicios utilizados' desde el archivo .bot
    var botConfig = BotConfiguration.Load(botFilePath, botFileSecret);
    services.AddSingleton(sp => botConfig ?? throw new InvalidOperationException(
        $"El archivo de configuracion .bot no pudo ser cargado"));

    // Obtiene la localizacion por defecto de appsettings.json
    var defaultLocale = Configuration.GetSection("defaultLocale").Get<string>();

    No implementado. AppInsights

    // Inicializa los 'servicios utilizados' y los añade como un singleton.
    //Estos servicio son incrustados mediante inyeccion de dependencias.
    var connectedServices = new BotServices(botConfig);
    services.AddSingleton(sp => connectedServices);

    No Implementado. Almacenamiento CosmosDB
    var datastore = new MemoryStorage();

    var userState = new UserState(datastore);
    var conversationState = new ConversationState(dataStore);

    services.AddSingleton(datastore);
    services.AddSingleton(userState);
    services.AddSingleton(conversationState);
    services.AddSingleton(new BotStateSet(userState, conversationState));
}

```

Startup 2.- Configuración de servicios utilizados por el Bot – ConfigureServices Parte 1

1. Declara e inicializa dos variables, que cargan información de *Configuration*: *botFilePath* que contiene la ruta al archivo de configuración de los servicios utilizados por el Bot (como el hospedaje local, el hospedaje Web, cognitivos, telemetría, transcripciones, ... la mayoría ubicados en la nube) y *botFileSecret* que contiene el secreto de encriptación de dicho archivo (en este caso vacío porque el objetivo es el *desarrollo* local no la *producción*).
2. La variable *botConfig* carga (o desencripta y carga) los servicios utilizados por el Bot y a continuación la añade a la colección como un servicio *singleton*. El procedimiento que marca *AZURE BOT SERVICE 4.2*, en primer lugar, carga los servicios y su configuración, para más tarde inicializarlos (paso 4).
3. La variable *defaultLocale* carga de *Configuration* la localización por defecto (es-ES).
4. Ahora, se crea una nueva variable *connectedServices* mediante la llamada al constructor de la clase *BotServices* pasando la variable *botConfig*, que contiene los servicios cargados y su configuración. La clase *BotServices* simplemente instancia cada servicio en cada una de sus propiedades de clase. Ahora, *connectedServices* se añade a los servicios como un *singleton*, mediante *services.AddSingleton*, (*singleton* significa que cada servicio solo poseerá una instancia) y podrán ser utilizados por la lógica del Bot mediante inyección de dependencias.
5. Se declara la variable *dataStore* que carga el almacenamiento de estados (para el Usuario y la Conversación) utilizado. El almacenamiento utilizado es el almacenamiento en memoria, *MemoryStorage*, que se borra al cerrar la aplicación Bot.

Como el proyecto consiste en el desarrollo de un Bot no es necesario un almacenamiento persistente que está más orientado a la *producción*.

6. Las variables *userState* y *conversationState* cargan del almacenamiento, *dataStorage*, respectivamente el estado del Usuario y de la Conversación.
7. Tanto el almacenamiento, como el estado de Usuario y de la Conversación, se añaden como servicios *singleton*, *services.AddSingleton*, para poder ser utilizados mediante inyección de dependencias en la lógica del Bot tras su puesta en marcha.
8. Ahora se añade otro servicio *singleton*, que fija ambos estados (*BotStateSet*) del Bot a los valores recogidos en el almacenamiento (*MemoryStorage*). Según el estado guardado en el almacenamiento, el Bot comenzará por una parte o por otra (orientado a un Bot con almacenamiento persistente como, por ejemplo, *CosmosDB* de Azure). En nuestro caso el Bot comenzará siempre por el principio porque el almacenamiento en memoria se borra al cerrar la aplicación Bot.

```
// Añade el Bot y algunas opciones que utilizará
services.AddBot<BotAprendizajeC>(options =>
{
    // Carga la conexión con la BotWebApp de Azure (mediante el punto de conexión y la contraseña) o con el host local
    var environment = _isProduction ? "production" : "development";
    var service = botConfig.Services.FirstOrDefault(s => s.Type == ServiceTypes.Endpoint && s.Name == environment);
    if (!(service is EndpointService endpointService))
    {
        throw new InvalidOperationException($"The .bot file does not contain an endpoint with name '{environment}'.");
    }
    options.CredentialProvider = new SimpleCredentialProvider(endpointService.AppId, endpointService.AppPassword);

    No Implementado. Telemetria - AppInsights

    // Captura cualquier error que ocurra en un turno de conversación y lo muestra por pantalla
    options.OnTurnError = async (context, exception) =>
    {
        //telemetryClient.TrackException(exception);
        await exception.Show(context);
    };
}

No Implementado. Transcripciones - BlobStorage

// Muestra el trabajo de escritura del bot en el Middleware (cuando está respondiendo/trabajando)
options.Middleware.Add(new ShowTypingMiddleware());

// Localización en el Middleware (fija la localidad en los servicios de Azure)
options.Middleware.Add(new SetLocaleMiddleware(defaultLocale ?? "en-us"));

// Autoguardado del estado en el Middleware (guarda el estado del bot y del usuario después de cada turno)
options.Middleware.Add(new AutoSaveStateMiddleware(userState, conversationstate));
});
```

Startup 3.- Configuración de servicios utilizados por el Bot – ConfigureServices Parte 2

9. ¡Ojo!, *services.AddBot* le dice al hospedaje Web que hay un servicio de tipo Bot... sin embargo, hasta ahora no se le ha dicho como tratar ese servicio. El método *ConfigureServices* terminará sabiendo que hay un servicio sin compilar. A parte, le da a entender que el punto de entrada a dicho servicio Bot será por la clase *BotAprendizajeC.cs*.

```

public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    // Configura Application Insights
    //_loggerFactory.AddApplicationInsights(app.ApplicationServices, LogLevel.Information);

    if (env.IsDevelopment())
    {
        //Capturador de errores sincronos y asincronos
        app.UseDeveloperExceptionPage();
    }

    app.UseDefaultFiles()
        //.UseBotApplicationInsights()
        .UseStaticFiles()
        .UseBotFramework();
}

```

Startup 4.- Configuración de servicios complementarios - Configure

Una vez terminado el método *ConfigureServices* se pondrá en marcha el próximo método, es decir, *Configure* (*Startup 4.- Configuración de servicios complementarios - Configure*). El método recibe dos interfaces: *IApplicationBuilder* que permite configurar algunos requisitos extra para la aplicación y *IHostingEnvironment* que proporciona el ambiente donde se mueve la aplicación, en este caso en *desarrollo*.

10. Si el ambiente está en *desarrollo* la aplicación podrá usar el capturador de errores síncronos y asíncronos (*app.UseDeveloperExceptionPage*).
11. *UseDefaultFiles* y *UseStaticFiles*, le dicen a la aplicación que puede usar todos los archivos y archivos estáticos que hay en la ruta raíz.
12. *UseBotFramework* hace que la aplicación entienda y pueda usar el servicio Bot que se encuentra en el método *ConfigureServices*.

¡Ahora sí! el programa irá directamente a compilar el Bot, es decir, se moverá a *services.AddBot* de *ConfigureServices* (*Startup 3.- Configuración de servicios utilizados por el Bot – ConfigureServices Parte 2*). La variable *options* cargará toda la configuración del adaptador como el *middleware*, el control de errores, etc que se activará en cada turno de conversación.

13. El primer bloque de código se encarga de la autenticación del Bot. En caso de estar en *producción* se comunicará con la *Web App Bot* correspondiente, por medio del identificador (*AppId*) y la contraseña (*AppPassword*), de la plataforma Microsoft Azure. En caso de estar en *desarrollo* no se comunicará con la plataforma Microsoft Azure y no será necesario ni el identificador ni la contraseña.
14. *OnTurnError* captura los errores que ocurran en un turno de conversación (pasada la puesta en marcha) y los muestra por pantalla, dando información detallada de algunos e información general de los demás.
15. Las tres últimas líneas comprenden al *middleware* utilizado que se ejecuta en cada turno de conversación:
 - o *ShowTypingMiddleware*: herramienta del *middleware* que indica el procesamiento/trabajo del Bot. En indicador se inicia tras la entrada de un

mensaje, se mantiene durante el trabajo de respuesta del Bot y finaliza cuando se envía el mensaje de salida (respuesta).

- *SetLocaleMiddleware*: herramienta del *middleware* que obtiene la localización del Usuario y configura el Bot para que envié mensajes de salida en el idioma correspondiente. En *producción* se obtendrá de la plataforma Microsoft Azure y en *desarrollo* será la localización por defecto, *defaultLocale* (es-ES).
- *AutoSaveStateMiddleware*: herramienta del *middleware* que guarda automáticamente el estado del Conversación y del Usuario en cada turno de conversación. No es efectiva cuando los estados deben ser guardados de forma manual.

Una vez finalizada la clase Startup.cs habrá finalizado la puesta en marcha de la aplicación Web o más concretamente la aplicación Bot. Así, el Bot estará listo para recibir mensajes de entrada.

3.2. Middleware

Tras la Puesta en marcha de la Aplicación Bot, el Bot se encuentra en disposición de aceptar mensajes (actividades) cuya entrada será por los métodos controladores de turnos, **OnTurnAsync**.

En primer lugar, la preferencia la tiene el *middleware* llamando a los tres métodos correspondientes (que derivan la interfaz *middleware*, *IMiddleware*): *ShowTypingMiddleware*, *SetLocaleMiddleware* y *AutoSaveStateMiddleware*; uno tras otro mediante delegados *next* como muestra la imagen de uno de ello, Middleware 1.- Clase SetLocaleMiddleware.

```
public class SetLocaleMiddleware : IMiddleware
{
    private readonly string _defaultLocale;

    public SetLocaleMiddleware(string defaultDefaultLocale)
    {
        _defaultLocale = defaultDefaultLocale;
    }

    public async Task OnTurnAsync(ITurnContext context, NextDelegate next, CancellationToken cancellationToken = default(CancellationToken))
    {
        var localeWebApp = context.Activity.Locale;
        var cultureInfo = string.IsNullOrEmpty(localeWebApp) ? new CultureInfo(_defaultLocale) : new CultureInfo(localeWebApp);

        CultureInfo.CurrentCulture = CultureInfo.CurrentCulture = cultureInfo;

        await next(cancellationToken).ConfigureAwait(false);
    }
}
```

Middleware 1.- Clase SetLocaleMiddleware

Para, por último, pasar el control a la clase derivada de la interfaz *IBot*, en nuestro caso, **BotAprendizajeC.cs** que será el punto de entrada al desempeño del Bot o punto de entrada de mensajería.

Nota: Todos los métodos OnTurnAsync reciben el contexto del turno (ITurnContext), el token de cancelación (CancellationToken) y, en ocasiones, el delegado al siguiente método OnTurnAsync (NextDelegate).

3.3.Punto de entrada de mensajería

Tras la delegación del ultimo método *OnTurnAsync* del *middleware* se da paso al punto de entrada de mensajería. Antes de dar paso a la conversación Usuario-Bot en sí, el Bot inicializa el constructor de la clase **BotAprendizajeC.cs**.

```
/// <summary> (Main) Punto de entrada principal del Bot.
public class BotAprendizajeC : IBot
{
    private readonly BotServices _services;
    private readonly ConversationState _conversationState;
    private readonly UserState _userState;
    private readonly IBotTelemetryClient _telemetryClient;

    private DialogSet _dialogs;

    /// <summary> Inicializa una nueva instancia de clase BotAprendizajeC.
    public BotAprendizajeC(BotServices botServices, ConversationState conversationState, UserState userState,
        IBotTelemetryClient telemetryClient = null)
    {
        _services = botServices ?? throw new ArgumentNullException(nameof(botServices));
        _conversationState = conversationState ?? throw new ArgumentNullException(nameof(conversationState));
        _userState = userState ?? throw new ArgumentNullException(nameof(userState));
        //_telemetryClient = telemetryClient ?? throw new ArgumentNullException(nameof(telemetryClient));

        _dialogs = new DialogSet(_conversationState.CreateProperty<DialogState>(nameof(BotAprendizajeC)));
        _dialogs.Add(new MainDialog(_services, _conversationState, _userState, _telemetryClient));
    }
}
```

BotAprendizajeC 1.- Campos y constructor

Según se muestra en *BotAprendizajeC 1.- Campos y constructor*:

El constructor posee un funcionamiento especial, acepta como parámetro cualquier servicio añadido (*services.AddSingleton*) en el método *ConfigureServices* de la clase **Startup.cs**. Esto se consigue gracias a la denominada *inyección de dependencias*.

- Las primeras tres líneas inicializan los tres campos de la clase y a la misma vez verifican que los tres parámetros del constructor no son nulos.
- *DialogSet* fija el diálogo del Bot. Este diálogo se considera “El diálogo” del que podrán emanar diferentes temáticas, en este caso, será una única temática dedicada al APRENDIZAJE DE C. Se fija pasándole el descriptor de acceso al estado del diálogo, *IStatePropertyAccessor<DialogState>*. El descriptor de acceso se crea a partir:
 - Del estado de la Conversación (*ConversationState*).
 - De la clase estado del diálogo (*DialogState*), que se fijó en **Startup.cs** con la llamada a *BotStateSet* (*Startup 2.- Configuración de servicios utilizados por el Bot – ConfigureServices Parte 1*).
- Ahora se añade el diálogo principal, *MainDialog*, que contendrá la temática del APRENDIZAJE DE C y servirá como diálogo base de los demás diálogos. Los parámetros



que acepta son los servicios que utiliza el Bot, el estado de la Conversación y del Usuario. *MainDialog* es del tipo *diálogo de componentes*, *ComponentDialog*, porque hereda de la clase *RouterDialog.cs* que a su vez hereda de *ComponentDialog*.

Nota: Cuando se pone en marcha el Bot se produce una actualización en la conversación, *ConversationUpdate*. Más concretamente se producen dos: una cuando se une el Bot y otra cuando se une el canal (Usuario).

```
/// <summary> Utilizado en cada turno de conversacion. Entrada de mensajes (Acti ...
public async Task OnTurnAsync(ITurnContext turnContext, CancellationToken cancellationToken)
{
    No Implementado
    //Crea el 'contexto del dialogo' (dc) a partir del turnContext y _conversationState
    var dc = await _dialogs.CreateContextAsync(turnContext);

    if (dc.ActiveDialog != null)
    {
        var result = await dc.ContinueDialogAsync();
    }
    else
    {
        await dc.BeginDialogAsync(nameof(MainDialog));
    }
}
```

BotAprendizajeC 2.- Punto de entrada de mensajería

Si se produce una actualización en la conversación (*ConversationUpdate*), se recibe un mensaje de entrada del Usuario, se desencadena un evento, ... comienza el método **OnTurnAsync**, plasmado en la imagen BotAprendizajeC 2.- Punto de entrada de mensajería. Este método recibe el contexto del turno y el token de cancelación:

- El contexto del turno, *ITurnContext*, se crea con la llegada de la actividad de entrada (que puede ser un mensaje de entrada, una actualización de la conversación, un evento del sistema, ...) que a su vez la contiene; a aparte contiene el adaptador, el estado del turno, etc.
- El token de cancelación, *CancellationToken*, produce la cancelación del turno de conversación cuando ocurre un error de procesamiento o una interrupción. Solo puede manejar situaciones sincrónicas y asincrónicas.

Ahora se crea el contexto del diálogo, *DialogContext*, a partir del diálogo fijado (*DialogSet*) y del contexto del turno (*ITurnContext*).

- Si hay un diálogo activo (como un diálogo en varios pasos (*diálogo en cascada*), diálogo en espera, etc) el *DialogContext* llamará al método *ContinueDialogAsync* para continuar el diálogo por donde se quedó.
- En caso de no haber diálogos activos el *DialogContext* comenzará el *MainDialog* con la llamada al método *BeginDialogAsync* colocándolo así en la pila de diálogos.

3.3.1. Encauzamiento de los mensajes de entrada

Cualquier mensaje de entrada hacia el Bot acaba en *RouterDialog* que se encargará de encauzarlo según corresponda.

```
/// <summary>
/// Clase derivada del Dialogo de Componentes <see cref="ComponentDialog"/>,
/// la cual sirve como encaminadora del dialogo principal <see cref="MainDialog"/>.
/// </summary>
public abstract class RouterDialog : ComponentDialog
{
    /// <summary> Constructor mediador entre MainDialog y ComponentDialog.
    public RouterDialog(string dialogId)
        : base(dialogId)
    {
    }

    /// <summary> COMENZADOR del RouterDialog. Metodo invalidado del ComponentDialog ...
    protected override Task<DialogTurnResult> OnBeginDialogAsync(DialogContext innerDc, object options,
    {
        return OnContinueDialogAsync(innerDc, cancellationToken);
    }
}
```

RouterDialog 1.- Constructor y método COMENZADOR

Según muestra la imagen RouterDialog 1.- Constructor y método COMENZADOR:

La llamada al método *BeginDialogAsync* para el *MainDialog* (BotAprendizajeC 2.- Punto de entrada de mensajería) buscará el método *OnBeginDialogAsync* anulado del *ComponentDialog* más próximo. Como el *MainDialog* no contiene ese método anulado entonces tendrá que buscarlo en su clase herencia, *RouterDialog*, que si lo contendrá.

El método *OnBeginDialogAsync* a su vez llamará directamente al método *OnContinueDialogAsync* del *RouterDialog*, anulado del *ComponentDialog*.

Cualquier método OnBeginDialogAsync o OnContinueDialogAsync , heredados de ComponentDialog, siempre aceptarán el contexto del diálogo (ContextDialog) y el token de cancelación (CancellationToken).

RouterDialog, o más concretamente el método OnContinueDialogAsync, se encargará de encaminar o encauzar los mensajes entrantes. Activar el mensaje de bienvenida, pasar el mensaje al MainDialog, ver si se ha pulsado un botón o continuar el diálogo activo.

```
/// <summary> CONTINUADOR del RouterDialog. Metodo invalidado del ComponentDialog ...
protected override async Task<DialogTurnResult> OnContinueDialogAsync(DialogContext innerDc, CancellationToken cancellationToken)
{
    var activity = innerDc.Context.Activity;

    // Muestra la tarjeta de Introduccion
    if (activity.IsStartActivity())
    {
        await OnStartAsync(innerDc);
    }

    switch (activity.Type)
    {
        case ActivityTypes.Message:
            {
                /// <remark>Boton Event: lanza un Evento a partir de una Accion recibida de un tarjeta.</remark>
                if (activity.Value!=null && activity.Value.ToString().Contains("Event") && innerDc.ActiveDialog==null)
                {
                    await OnEventAsync(innerDc);
                }
                /// <remark>Continua con el dialogo activo, sino ENCAMINA un nuevo dialogo.</remark>
                else if (!string.IsNullOrEmpty(activity.Text) || !string.IsNullOrEmpty(activity.Value.ToString()))
                {
                    if (string.IsNullOrEmpty(activity.Text))
                        innerDc.Context.Activity.Text = ((dynamic)activity.Value).action.ToString();

                    var result = await innerDc.ContinueDialogAsync(); //DialogTurnResult (status, Result)
                }
            }
    }
}
```

RouterDialog 2.- Método CONTINUADOR Parte 1

```
switch (result.Status)
{
    case DialogTurnStatus.Empty:
        {
            await ReconocimientoExpresionesAsync(innerDc);
            break;
        }
    case DialogTurnStatus.Complete:
        {
            await CompleteAsync(innerDc);
            // "Seguridad adicional" en la finalizacion del dialogo
            await innerDc.EndDialogAsync();
            break;
        }
    default:
        break;
}
break;
}

case ActivityTypes.Event:
//await OnEventAsync(innerDc);
break;
default:
//await OnSystemMessageAsync(innerDc);
break;
}
return EndOfTurn;
}
```

RouterDialog 3.- Método CONTINUADOR Parte 2

Mensaje de bienvenida

Las primeras líneas de la imagen RouterDialog 2.- Método CONTINUADOR Parte 1 corresponden a la ejecución del *mensaje de bienvenida* o *Tarjeta de Introducción*.

```
/// <summary>
/// Método que comprueba las actualizaciones en la conversación,
/// para cada canal.
/// </summary>
public static bool IsStartActivity(this Activity activity)
{
    switch (activity.ChannelId)
    {
        case Channels.Skype:
            {
                if (activity.Type == ActivityTypes.ContactRelationUpdate && activity.Action == "add")
                    return true;
                return false;
            }
        case Channels.Directline:
        case Channels.Emulator:
        case Channels.Webchat:
        case Channels.Msteams:
            {
                // Cuando el Bot se une a la conversación y hay una actualización en la conversación
                if (activity.Type == ActivityTypes.ConversationUpdate)
                    if (activity.MembersAdded.Any(m => m.Id == activity.Recipient.Id))
                        return true;
                return false;
            }
        default:
            return false;
    }
}
```

Código 2.- Método creado IsStartActivity

El método extensión de *Activity*, *IsStartActivity*, comprueba si ha habido una actualización en la conversación y da soporte a varios canales según la imagen Código 2.- Método creado *IsStartActivity*.

Para el emulador (**Bot Framework Emulator**), si la actividad de entrada es una actualización de la conversación, *ConversationUpdate*, y es el Bot el que se une a la conversación, el método creado *IsStartActivity* devolverá *True* y se desencadenará el método *OnStartAsync* creado.

```
/// <summary> Método utilizado cuando hay una ConversationUpdate.
protected override async Task OnStartAsync(DialogContext innerDc, CancellationToken cancellationToken)
{
    await _responder.ReplyWith(innerDc.Context, MainResponses.ResponseIds.Intro);
}
```

MainDialog 1.- Mensaje de bienvenida

El método *OnStartAsync* está ubicado en el *MainDialog* (MainDialog 1.- Mensaje de bienvenida) y contiene una única línea de código que, a groso modo, envía una actividad (mensaje de salida) de respuesta hacia el Usuario que contendrá el *mensaje de bienvenida* o *Tarjeta de Introducción*.

`_responder` se comporta como un gestor de respuestas (plantillas) y su método asociado `ReplyWith` se encarga de enviar la respuesta (plantilla), con identificador `Intro`, hacia el Usuario.

Después de esto finalizará el método `OnStartAsync` y se devolverá el control al método `OnContinueDialogAsync` de `RouterDialog` de la imagen RouterDialog 2.- Método CONTINUADOR Parte 1.

A continuación, se comprobará el tipo de actividad en el selector (`switch`). La actividad es de tipo `ConversationUpdate` por lo tanto se ejecutará la opción por defecto que romperá el `switch` según se muestra en RouterDialog 3.- Método CONTINUADOR Parte 2. Por último, se ejecuta `EndOfTurn` que marcará el final del turno del Bot.

Mensaje de entrada

Un mensaje de entrada, según AZURE BOT SERVICE 4.2, se considera tanto texto escrito como la acción de un botón por el Usuario. Ambas se almacenan en la actividad de entrada, respectivamente en `Activity.Text` y `Activity.Value`, y nunca podrán ir juntas porque son mutuamente excluyentes.

A su vez, los botones contienen texto específico que actúa sobre el código del Bot. Se distinguen dos tipos según los patrones seguidos para la construcción del Bot:

- **Botones normales.** Contienen texto simple que será interpretado por `Luis` o mediante igualdad simple.
- **Botones especiales.** Contienen palabras clave que activarán los eventos del Bot creados.

Aclarado lo anterior, cuando el Usuario envíe un mensaje de entrada y obviando la parte del punto de entrada de mensajería (BotAprendizajeC 2.- Punto de entrada de mensajería) el desencadenamiento del código se sitúa en RouterDialog 2.- Método CONTINUADOR Parte 1, concretamente en el método `OnContinueDialogAsync`. Aquí se verificará que el tipo de actividad es de mensaje, `ActivityTypes.Message`. A continuación, se comprobará si se ha accionado un botón especial, o si se ha escrito o pulsado un botón normal.

Botón especial

La acción de un botón está contenida en `Activity.Value`.

Habiendo verificado que la actividad es de tipo mensaje, `ActivityTypes.Message`, se comprobará si la acción no es nula (`null`), contiene la palabra clave “Event” y no hay un diálogo activo según se muestra en RouterDialog 2.- Método CONTINUADOR Parte 1. Si es así la acción vendrá de un **botón especial** y se iniciará el método `OnEventAsync` credo.

Importante: Estos eventos no son eventos al uso según los define C#, sino que son eventos creados especialmente para cubrir las acciones de las tarjetas adaptativas creadas, es decir, se encargan de dar paso a todas las acciones de los botones de la Tarjeta de Introducción (mensaje de bienvenida) y de la Tarjeta de Ayuda.

```

/// <summary> Acciones (botones) de tarjeta.
protected override async Task OnEventAsync(DialogContext innerDc, CancellationToken cancellationToken = default(CancellationToken)
{
    var context = innerDc.Context;
    //Acciones de la tarjeta Intro
    if (context.Activity.Value != null && context.Activity.Value.ToString().Contains("Intro"))
    {
        switch (((dynamic)context.Activity.Value).action.ToString())
        {
            case "indiceTEMARIOIntroEvent":
                await _sharedTEMARIOResponder.ReplyWith(context, SharedTEMARIOResponses.ResponseIds.IndiceTEMARIO);
                await _sharedTEMARIOResponder.ReplyWith(context, SharedTEMARIOResponses.ResponseIds.SelecOEscri);
                break;
            case "comenzarTEMARIOIntroEvent":
                await innerDc.BeginDialogAsync(nameof(Cap01IntroduccionDialog));
                break;
            case "indicePROBLEMASIntroEvent":
                await _sharedPROBLEMASResponder.ReplyWith(context, SharedPROBLEMASResponses.ResponseIds.IndicePROBLEMAS);
                await _sharedPROBLEMASResponder.ReplyWith(context, SharedPROBLEMASResponses.ResponseIds.SelecOEscriPROBLEMAS);
                break;
            case "comenzarPROBLEMASIntroEvent":
                await innerDc.BeginDialogAsync(nameof(Problema01Dialog));
                break;
            case "aleatorioPROBLEMASIntroEvent":
                await innerDc.BeginDialogAsync(Functions.DialogoAleatorio(MainStrings.PROBLEMAS_PATH));
                break;
            case "biblioAgradeciIntroEvent":
                await _responder.ReplyWith(context, MainResponses.ResponseIds.BiblioAgradeci);
                await innerDc.CancelAllDialogsAsync();
                break;
            case "No Implementados":
                break;
        }
    }
}

```

MainDialog 2.- Acciones de tarjeta o eventos Parte 1

```

//Acciones de la tarjeta Help
else if (context.Activity.Value != null && context.Activity.Value.ToString().Contains("Help"))
{
    switch (((dynamic)context.Activity.Value).action.ToString())
    {
        case "InfoTEMARIOHelpEvent":
            await _helpResponder.ReplyWith(context, HelpResponses.ResponseIds.InfoTEMARIO);
            break;
        case "InfoPROBLEMASHelpEvent":
            await _helpResponder.ReplyWith(context, HelpResponses.ResponseIds.InfoPROBLEMAS);
            break;
        case "BusqEnTEMARIOHelpEvent":
            await _helpResponder.ReplyWith(context, HelpResponses.ResponseIds.BusqEnTEMARIO);
            break;
        case "BusqEnPROBLEMASHelpEvent":
            await _helpResponder.ReplyWith(context, HelpResponses.ResponseIds.BusqEnPROBLEMAS);
            break;
        case "DifeEjerProbleHelpEvent":
            await _helpResponder.ReplyWith(context, HelpResponses.ResponseIds.DifeEjerProble);
            break;
        case "AlmacePuntosConsulHelpEvent":
            await _helpResponder.ReplyWith(context, HelpResponses.ResponseIds.AlmacePuntosConsul);
            break;
    }
}

```

MainDialog 3.- Acciones de tarjeta o eventos Parte 2

Según muestran ambas imágenes MainDialog 2.- Acciones de tarjeta o eventos Parte 1 y MainDialog 3.- Acciones de tarjeta o eventos Parte 2, `OnEventAsync` recoge todas las acciones de la *Tarjeta de Introducción (mensaje de bienvenida)* y de la *Tarjeta de Ayuda*. Cada acción proporcionará una respuesta o respuestas (ReplyWith) (desde un gestor de respuestas `_Responder`) con un identificador de respuesta) o comenzará un diálogo (BeginDialogAsync).

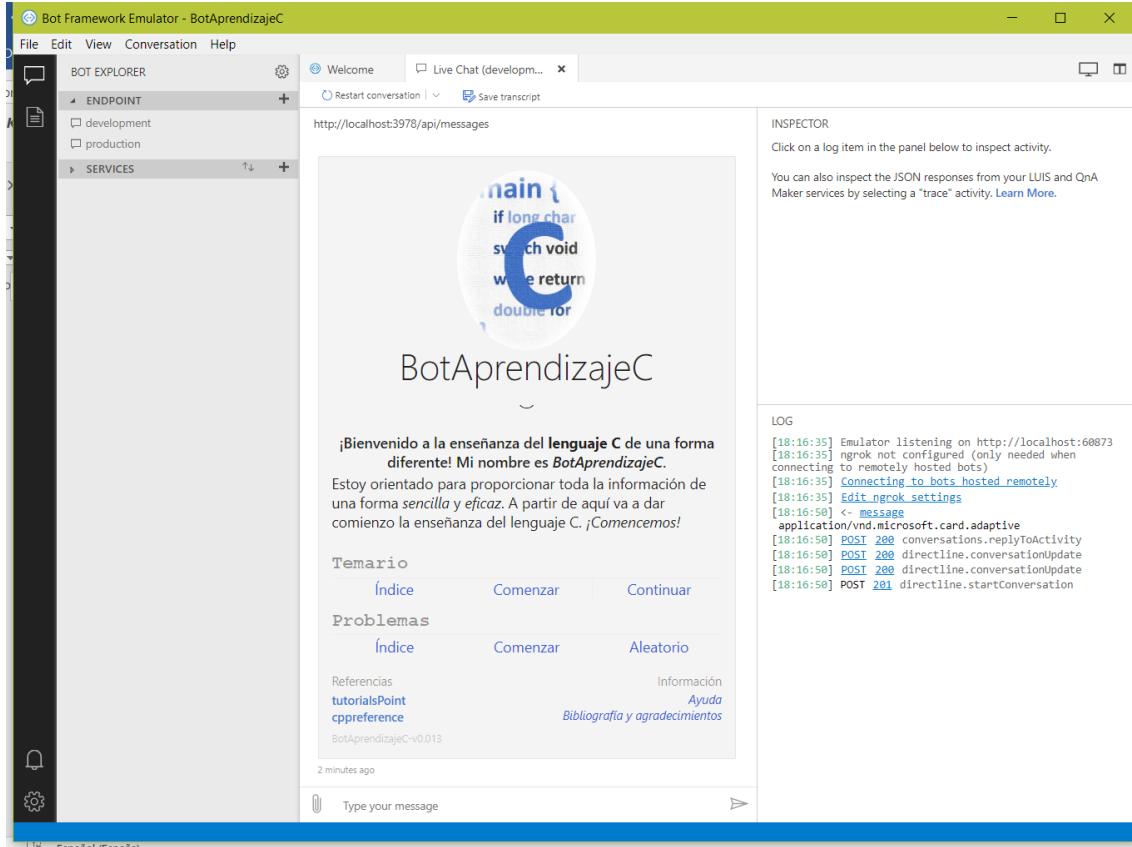
Una vez ejecutada la acción correspondiente terminará el método *OnEventAsync*, pasando el control de nuevo al método *OnContinueDialogAsync* (RouterDialog 2.- Método CONTINUADOR Parte 1) y ejecutando *EndOfTurn* (RouterDialog 3.- Método CONTINUADOR Parte 2) que marcará el final del turno de Bot.

Importante: El uso de variables o casteos dinámicos, *dynamic*, está restringido al acceso a miembros dinámicos de la propiedad *Value*, *Activity.Value*, de una actividad de entrada.

Para el acceso al miembro dinámico “action” por ejemplo se puede hacer así:

((dynamic)Activity.Value).action.

El método creado *OnEventAsync* da soporte a los botones especiales pulsados en la *Tarjeta de Introducción* o en la *Tarjeta de Ayuda*.



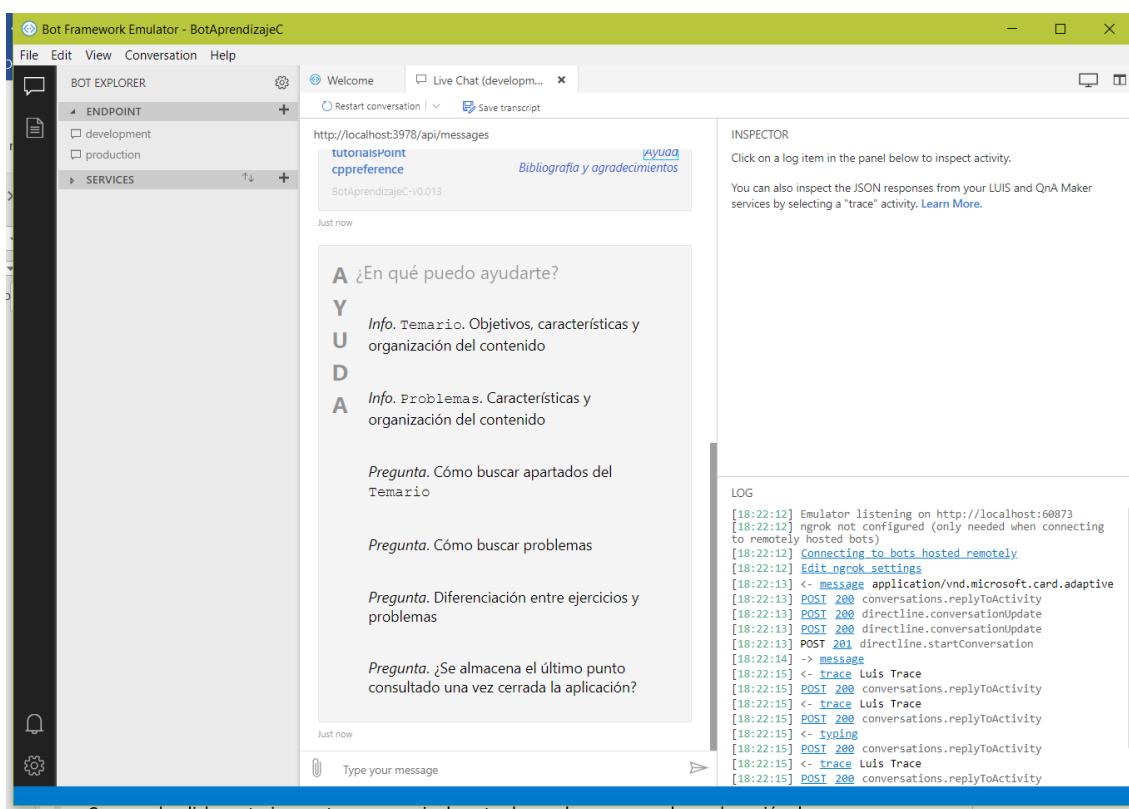
Bot Framework Emulator 1.- Tarjeta de Introducción

Según muestra la imagen Bot Framework Emulator 1.- Tarjeta de Introducción:

Los **botones especiales** para la *Tarjeta de Introducción* son Índice y Comenzar del TEMARIO; Índice, Comenzar y Aleatorio de los PROBLEMAS; y Bibliografía y agradecimientos. Los demás botones como Continuar, tutorialsPoint, cppreference y Ayuda se consideran **botones normales**.

- TEMARIO.

- **Índice.** Muestra el índice del temario dividido en capítulos y apartados de capítulo, subapartados y ejercicios. Además, proporciona información sencilla de como buscar un apartado en concreto.
- **Comenzar.** Comienza el TEMARIO por el primer apartado del capítulo 1.
- **Continuar.** Continua por el último punto consultado del APRENDIZAJE DE C. En caso de no haber consultado uno se lanza un mensaje de error.
- **PROBLEMAS.**
 - **Índice.** Muestra el índice de los problemas dividido en capítulos y problemas por capítulo. Además, proporciona información sencilla sobre como buscar un problema.
 - **Comenzar.** Comienza los PROBLEMAS por el primer problema del capítulo 1.
 - **Aleatorio.** Lanza un problema aleatorio.
- **Ayuda.** Proporciona acceso a la *Tarjeta de Ayuda*.
- **Bibliografía y agradecimientos.** Muestra toda la información sobre la bibliografía utilizada para la realización del Bot y agradece a las partes implicadas, libros, páginas Web, ... su colaboración.
- **Enlace Web (tutorialsPoint).** Enlace Web que contiene referencias a C.
- **Enlace Web (cppreference).** Enlace Web que contiene referencias a C.



Bot Framework Emulator 2.- Tarjeta de Ayuda

Según muestra la imagen Bot Framework Emulator 2.- Tarjeta de Ayuda:

Los **botones especiales** para la *Tarjeta de Ayuda* son seis en total; dos informativos y cuatro preguntas.

- **Info.** TEMARIO. **Objetivos, características y organización del contenido.** Muestra toda la información relacionada con el TEMARIO como la orientación y objetivos, características y organización del contenido.
- **Info.** PROBLEMAS. **Características y organización del contenido.** Muestra toda la información relacionada con los PROBLEMAS como las características y la organización del contenido.
- **Pregunta. Cómo buscar apartados del TEMARIO.** Muestra información detallada sobre como buscar un apartado del TEMARIO además de mostrar algunos ejemplos de funcionamiento.
- **Pregunta. Como buscar problemas.** Muestra información detallada sobre como buscar un problema de los PROBLEMAS además de mostrar algunos ejemplos de funcionamiento.
- **Pregunta. Diferenciación entre ejercicios y problemas.** Describe de manera precisa la diferencia que existe entre los ejercicios incluidos en el TEMARIO y problemas incluidos en los PROBLEMAS.
- **Pregunta. ¿Se almacena el último punto consultado una vez cerrada la aplicación?** Da respuesta a la pregunta propuesta.

Escritura o botón normal

Como se ha dicho anteriormente un mensaje de entrada puede ser generado por la acción de un botón (*Activity.Value*) o por texto escrito (*Activity.Text*).

Según se muestra en RouterDialog 2.- Método CONTINUADOR Parte 1.

El Bot ha sido construido para que la propiedad *Text* de una actividad de entrada nunca sea nula (*null*). Es decir, en caso de serlo, se ejecutarán las líneas de código correspondientes que pasarán ‘el texto de la acción dinámica del **botón normal**’ (*(dynamic)activity.Value.action.ToString()*) a la propiedad *Text*.

- Si hay un diálogo activo como un diálogo sobre un punto del APRENDIZAJE DE C, se continuará por él. Cada apartado del TEMARIO o problema de los PROBLEMAS está incrustado en un *diálogo en cascada*. Según la situación en la que se encuentre el diálogo activo el método *ContinueDialogAsync* devolverá un tipo de estado:

- **Continuación.** Un paso de un *diálogo en cascada* termina con un *diálogo de petición*, pasando el control de nuevo al método *ContinueDialogAsync*, devolviendo un estado del diálogo “en espera”, *DialogTurnStatus.Waiting*, y finalizando el turno del Bot, *EndOfTurn*.
- **Cancelación.** Si el diálogo activo es cancelado, se pasa el control de nuevo al método *ContinueDialogAsync*, devolviendo un estado del diálogo “completado”, *DialogTurnStatus.Complete*, y lanzando el método *CompleteAsync* (MainDialog 4.- Diálogo completado).

```
protected override async Task CompleteAsync(DialogContext innerDc, CancellationToken cancellationToken)
{
    // El dialogo activo en la pila finalizo con estado completado
    await _responder.ReplyWith(innerDc.Context, MainResponses.ResponseIds.Completed);
}
```

MainDialog 4.- Diálogo completado

Este método simplemente envía una respuesta al usuario (*ReplyWith*) (desde un gestor de respuestas (*_responder*) con identificador *Completed*) invitándole a continuar. Más tarde, devuelve el control al método *OnContinueDialogAsync* de RouterDialog 3.- Método CONTINUADOR Parte 2 y finaliza el turno del Bot, *EndOfTurn*.

- **Reemplazo.** Cualquier diálogo se reemplazará siempre por otro *diálogo en cascada*. El diálogo activo reemplazado comienza por el primer paso de cascada hasta lanzar el *diálogo de petición*. En ese momento se pasa el control al método *ContinueDialogAsync*, devolviendo un estado del diálogo “en espera”, *DialogTurnStatus.Waiting*, y finalizando el turno del Bot, *EndOfTurn*.
- **Finalización.** Cuando se llegue al final del TEMARIO o de los PROBLEMAS, finalizará el diálogo activo con la llamada a método *EndDialogAsync*, devolviendo un estado “completado”, *DialogTurnStatus.Complete*. A partir de aquí, se realizarán los mismos pasos de la **Cancelación**.
- Si no hay un diálogo activo se devolverá un estado del diálogo “vacío”, *DialogTurnStatus.Empty*, y se iniciará la función *ReconocimientoExpresionesAsync* encargada de reconocer la intención (*Luis*) de la actividad de entrada (*Activity.Text*).

3.3.2. Main

Toda la información que recoge el Bot esta incrustada en la clase *MainDialog* y puede ser utilizada por sus métodos.

Campos y Constructor

```
public class MainDialog : RouterDialog
{
    private BotServices _services;
    //Administracion de estados
    private ConversationState _conversationState;
    private UserState _userState;
    private IStatePropertyAccessor<UserPropertiesState> _userPropertiesAccessor;
    //Respondedores
    private MainResponses _responder = new MainResponses();
    private SharedTEMARIOResponses _sharedTEMARIOResponder = new SharedTEMARIOResponses();
    private SharedPROBLEMASResponses _sharedPROBLEMASResponder = new SharedPROBLEMASResponses();
    private HelpResponses _helpResponder = new HelpResponses();

    public MainDialog(BotServices services, ConversationState conversationState, UserState userState,
        IBotTelemetryClient telemetryClient) : base(nameof(MainDialog))
    {
        _services = services ?? throw new ArgumentNullException(nameof(services));

        _conversationState = conversationState;
        _userState = userState;
        TelemetryClient = telemetryClient;

        _userPropertiesAccessor = _userState.CreateProperty<UserPropertiesState>(nameof(UserPropertiesState));

        AddDialog(new UserDataRequestDialog(_services,
            _userState.CreateProperty<UserDataRequestState>(nameof(UserDataRequestState)), telemetryClient));

        DialogosTEMARIO(); // Todo el contenido del Temario de C en 9 capítulos
        DialogosPROBLEMAS(); // Todo el contenido de los Problemas de C del capítulo 3 al 9
    }
}
```

MainDialog 5.- Campos y constructor

```

private void DialogosTEMARIO()
{
    #region CAP01
    AddDialog(new Cap01IntroduccionDialog(_services, _userPropertiesAccessor, TelemetryClient));
    #endregion
    CAP02
    CAP03
    CAP04
    CAP05
    CAP06
    CAP07
    CAP08
    CAP09
}
private void DialogosPROBLEMAS()
{
    CAP03_04
    CAP05
    CAP06
    CAP07
    CAP08
    CAP09
}

```

MainDialog 6.- Diálogos del TEMARIO y los PROBLEMAS

Los campos y el constructor, según se muestra en la imagen MainDialog 5.- Campos y constructor, contienen los servicios (modelos **Luis**), el estado de la Conversación y del Usuario, los descriptores de acceso a las propiedades del Usuario, los diferentes gestores de respuestas (**_responder**) y los métodos que cargan todos los diálogos sobre los puntos del APRENDIZAJE DE C.

Todos los diálogos sobre los puntos del APRENDIZAJE DE C necesitan los mismos parámetros como los servicios y los descriptores de acceso a las propiedades de Usuario, según lo muestra el primer punto de TEMARIO en MainDialog 6.- Diálogos del TEMARIO y los PROBLEMAS.

Nota: El comando de telemetría, TelemetryClient, no se explica porque el Bot está en desarrollo. La telemetría está diseñada para medir la actividad del Bot ante los Usuarios externos, por lo tanto, no es necesaria para desarrollar el Bot.

Reconocimiento del mensaje de entrada

La estructura utilizada para el reconocimiento de expresiones, **Activity.Text**, consta de un **modelo distribuidor**, que analiza la expresión y la distribuye a uno de los **modelos principales**. Uno de los **modelos principales** analiza de nuevo la expresión y se encarga de enviar una respuesta o respuestas, o comenzar un apartado del APRENDIZAJE DE C.

```

/// <summary> Método utilizado para reconocer el mensaje de entrada.</summary>
protected override async Task ReconocimientoExpresionesAsync(DialogContext innerDc, CancellationToken cancellationToken = default)
{
    /// <remarks>Modelo Luis "Dispatch" es un modelo cuyos intentos son otros modelos (Luis, Qna, ...).</remarks>
    var dispatchResult = await _services.DispatchRecognizer.RecognizeAsync<Dispatch>(innerDc, true, cancellationToken.None);
    var intent = dispatchResult?.TopIntent().intent;
}

```

MainDialog 7.- Método ReconocimientoExpresionesAsync Parte 1

El método encargado del reconocimiento del mensaje de entrada, *Activity.Text*, es *ReconocimientoExpresionesAsync*. El reconocimiento se realiza mediante modelos **LUIS**. Las primeras dos líneas deciden, mediante el *modelo distribuidor*, **Dispatch**, que *modelo principal* utilizar, como se muestra en MainDialog 7.- Método ReconocimientoExpresionesAsync Parte 1.

El modelo **Dispatch** contiene todas las expresiones (*expressions*) de los *modelos principales*, agrupadas en una intención (*intent*) para cada modelo. Cuando una expresión (mensaje de entrada) corresponde con un *modelo principal*, **LUIS** devuelve una intención que corresponde a dicho modelo. Hay tres *modelos principales* disponibles: **General**, **TemarioC** y **ProblemasC**.

General

```
/// <remarks>Modelo Luis "general" recoge las acciones basicas del usuario.</remarks>
if (intent == Dispatch.Intent.l_general)
{
    var generalRecognizer = _services.GeneralRecognizer;

    if (generalRecognizer == null)
        throw new Exception("El modelo LUIS especificado no pudo ser encontrado en la configuracion 'BotServices'.");
    else
    {
        var result = await generalRecognizer.RecognizeAsync<General>(innerDc, true, CancellationToken.None);
        var generalIntent = result?.TopIntent().intent;
```

MainDialog 8.- Método ReconocimientoExpresionesAsync Parte 2

```
// Comutador de intentos
switch (generalIntent)
{
    case General.Intent.Bibliography:
        await _responder.ReplyWith(innerDc.Context, MainResponses.ResponseIds.BiblioAgradeci);
        break;
    case General.Intent.CreateNewUser:
        await innerDc.BeginDialogAsync(nameof(UserDataRequestDialog));
        break;
    case General.Intent.Help: // Muestra la tarjeta de Ayuda
        await _helpResponder.ReplyWith(innerDc.Context, HelpResponses.ResponseIds.Help);
        break;
    case General.Intent.Intro: // Muestra la tarjeta de Introduccion
        await _responder.ReplyWith(innerDc.Context, MainResponses.ResponseIds.Intro);
        break;
    case General.Intent.ShowPrevious:
        var state = await _userPropertiesAccessor.GetAsync(innerDc.Context, () => new UserPropertiesState());
        if (string.IsNullOrEmpty(state.UltimoApartado)) {
            await _responder.ReplyWith(innerDc.Context, MainResponses.ResponseIds.Negacion_MostrarAnterior);
            break;
        }
        await innerDc.BeginDialogAsync(state.UltimoApartado);
        break;
    case General.Intent.None:
    default: // Send confused message
        await _responder.ReplyWith(innerDc.Context, MainResponses.ResponseIds.Confused);
        break;
}
```

MainDialog 9.- Método ReconocimientoExpresionesAsync Parte 3

El modelo **General** recoge el funcionamiento básico del Bot como la Bibliografía, nuevo Usuario, Ayuda, Introducción o Menú y mostrar el último apartado consultado.

Según muestran las imágenes MainDialog 8.- Método ReconocimientoExpresionesAsync Parte 2 y MainDialog 9.- Método ReconocimientoExpresionesAsync Parte 3. Una vez obtenido el resultado para el mensaje de entrada mediante el método *RecognizeAsync* del modelo **General** y la intención a partir del resultado, se actuará según convenga. Para la intención...

- *Bibliography*. Muestra la *Tarjeta de la Bibliografía (ReplyWith)* como respuesta de un gestor de respuestas (*_responder*) con identificador *BiblioAgradeci*.
- *CreateNewUser*. Comienza un diálogo nuevo (*BeginDialogAsync*) concretamente un *diálogo en cascada* derivado de *AprendizajeCDialog.cs* que pide al usuario su nombre localización y e-mail. Este diálogo servirá como base para una posible autenticación del Usuario en versiones posteriores.
- *Help*. Muestra la *Tarjeta de Ayuda (ReplyWith)* como respuesta de un almacén de respuesta (*_helpResponder*) con identificador *Help*.
- *ShowPrevious*. Muestra, de nuevo, el último apartado consultado del *TEMARIO* o de los *PROBLEMAS*. En caso de no haber consultado uno se envía una respuesta negativa (*ReplyWith*) de un gestor de respuestas (*_responder*) con identificador *Negacion_MostrarAnterior*.
- *None* o *default*. Envía una respuesta de confusión (*ReplyWith*) de un gestor de respuestas (*_responder*) con identificador *Confused*.

TemarioC

```
/// <remarks>Modelo Luis "temarioC" que muestra el contenido del temario de C.</remarks>
else if (intent == Dispatch.Intent.l_temarioC)
{
    var temarioCRecognizer = _services.TemarioCRecognizer;

    if (temarioCRecognizer == null)
        throw new Exception("El modelo LUIS especificado no pudo ser encontrado en la configuracion 'BotServices'.");
    else
    {
        var temarioCResult = await temarioCRecognizer.RecognizeAsync<TemarioC>(innerDc, true, CancellationToken.None);
        var temarioCIntent = temarioCResult?.TopIntent().intent;

        if(temarioCResult?.TopIntent().score > 0.4)
        {
            //responde con respuesta confusa, con el indice o muestra el dialogo correspondiente
            if (temarioCIntent == TemarioC.Intent.None)
                await _responder.ReplyWith(innerDc.Context, MainResponses.ResponseIds.Confused);
            else if (temarioCIntent == TemarioC.Intent.IndiceTEMARIO)
            {
                await _sharedTEMARIOResponder.ReplyWith(innerDc.Context, SharedTEMARIOResponses.ResponseIds.IndiceTEMARIO);
                await _sharedTEMARIOResponder.ReplyWith(innerDc.Context, SharedTEMARIOResponses.ResponseIds.SelecOEscri);
            }
            else
            {
                var intentDialog = temarioCIntent.ToString() + "Dialog";
                await innerDc.BeginDialogAsync(intentDialog);
            }
        }
        else
            await _responder.ReplyWith(innerDc.Context, MainResponses.ResponseIds.Confused);
    }
}
```

MainDialog 10.- Método ReconocimientoExpresionesAsync Parte 4

El modelo **TemarioC** contiene todas las expresiones agrupadas en intenciones que lanzan los apartados del TEMARIO. Las expresiones son frases que denominan el apartado de un capítulo y pueden ser del tipo “capítulo (nº) (apartado)” o palabras clave como “if”, “for”, “switch”, etc.

Según muestra la imagen MainDialog 10.- Método ReconocimientoExpresionesAsync Parte 4, una vez obtenido el resultado para el mensaje de entrada mediante el método *RecognizeAsync* del modelo **TemarioC** y la intención a partir del resultado, en primer lugar, se comprobará que posee una puntuación apta y, en segundo lugar, si es:

- *None*. Envía una respuesta de confusión (*ReplyWith*) de un gestor de respuestas (*_responder*) con identificador *Confused*.
- *IndiceTEMARIO*. Muestra el carrusel de tarjetas correspondiente a cada capítulo del TEMARIO y una respuesta del funcionamiento de búsqueda, mediante dos plantillas (*ReplyWith*) de un almacén de plantillas (*_sharedTEMARIOResponses*) con identificador *IndiceTEMARIO* y *SelecOEscribir*.
- Apartados del TEMARIO. La opción por defecto comenzará un diálogo (*BeginDialogAsync*) del TEMARIO. Se ha construido de tal forma que, el nombre del diálogo coincidirá con la intención proporcionada por *Luis* + “Dialog”.

Si la puntuación no es apta se enviará una respuesta de confusión (*ReplyWith*) de un gestor de respuestas (*_responder*) con identificador *Confused*.

ProblemasC

```
//<remarks>Modelo Luis "problemasC" que muestra el contenido de los problemas de C.</remarks>
else if (intent == Dispatch.Intent.l_problemasC)
{
    var problemasCRecognizer = _services.ProblemasCRecognizer;

    if (problemasCRecognizer == null)
        throw new Exception("El modelo LUIS especificado no pudo ser encontrado en la configuracion 'BotServices'.");
    else
    {
        var problemasCResult = await problemasCRecognizer.RecognizeAsync<ProblemasC>(innerDc, true, CancellationToken.None);
        var problemasCIntent = problemasCResult?.TopIntent().intent;

        if (problemasCResult?.TopIntent().score > 0.4)
        {
            //responde con respuesta confusa, con el indice o muestra el dialogo correspondiente
            if (problemasCIntent == ProblemasC.Intent.None)
                await _responder.ReplyWith(innerDc.Context, MainResponses.ResponseIds.Confused);
            else if (problemasCIntent == ProblemasC.Intent.IndicePROBLEMAS)
            {
                await _sharedPROBLEMASResponder.ReplyWith(innerDc.Context, SharedPROBLEMASResponses.ResponseIds.IndicePROBLEMAS);
                await _sharedPROBLEMASResponder.ReplyWith(innerDc.Context, SharedPROBLEMASResponses.ResponseIds.SelecOEscriPROBLEMAS);
            }
            else
            {
                var intentDialog = problemasCIntent.ToString() + "Dialog";
                await innerDc.BeginDialogAsync(intentDialog);
            }
        }
        else
            await _responder.ReplyWith(innerDc.Context, MainResponses.ResponseIds.Confused);
    }
}
else // Si el modelo Dispatch no encuentra el intento, envia una respuesta "confused".
    await _responder.ReplyWith(innerDc.Context, MainResponses.ResponseIds.Confused);
```

MainDialog 11.- Método ReconocimientoExpresionesAsync Parte 5

El modelo **ProblemasC** contiene todas las expresiones agrupadas en intenciones que lanzan cada problema de los PROBLEMAS. Las expresiones son frases que denominan un problema, son del tipo “problema (nº)”.

Según muestra la imagen MainDialog 11.- Método ReconocimientoExpresionesAsync Parte 5.

Una vez obtenido el resultado para el mensaje de entrada mediante el método

RecognizeAsync del modelo **ProblemasC** y la intención a partir del resultado, en primer lugar, se comprobará que posee una puntuación apta y, en segundo lugar, si es:

- *None*. Envía una respuesta de confusión (*ReplyWith*) de un gestor de respuestas (*_responder*) con identificador *Confused*.
- *IndicePROBLEMAS*. Muestra el carrusel de tarjetas correspondiente a cada capítulo de los PROBLEMAS y una respuesta con el funcionamiento de búsqueda de un problema, mediante dos plantillas (*ReplyWith*) de un almacén de plantillas (*_sharedPROBLEMASResponses*) con identificador *IndicePROBLEMAS* y *SelecOEscribirPROBLEMAS*.
- Problema de los PROBLEMAS. La opción por defecto comenzará un diálogo (*BeginDialogAsync*) de los PROBLEMAS. Se ha construido de tal forma que el nombre del diálogo coincidirá con la intención proporcionada por **Luis** + “Dialog” .

Si la puntuación no es apta se enviará una respuesta de confusión (*ReplyWith*) de un gestor de respuestas (*_responder*) con identificador *Confused*.

3.4. Gestor de repuestas

Todas las respuestas que envía el Bot, como mensaje de salida, se tratan en gestores de respuestas (*_responder/TemplateManager*) que se complementan con *archivos de recursos* y los *archivos de notación*.

```
public class MainResponses : TemplateManager
{
    private static int MaxConfused = 0;
    private static LanguageTemplateDictionary _responseTemplates = new LanguageTemplateDictionary
    {
        ["default"] = new TemplateIdMap
        {
            { ResponseIds.Completed, (context, data) =>
                MessageFactory.Text(
                    text: MainStrings.COMPLETED,
                    ssml: MainStrings.COMPLETED,
                    inputHint: InputHints.AcceptingInput)
            },
            { ResponseIds.Confused, (context, data) => ConfusedResponses(context, data) },
            { ResponseIds.Negacion_MostrarAnterior, (context, data) =>
                MessageFactory.Text(
                    text: MainStrings.NEGACION_MOSTRARANTERIOR,
                    ssml: MainStrings.NEGACION_MOSTRARANTERIOR,
                    inputHint: InputHints.AcceptingInput)
            },
            { ResponseIds.Intro, (context, data) => BuildIntroCard(context, data) },
            { ResponseIds.BiblioAgradeci, (context, data) => BiblioAgradeciCard(context, data) }
        };
    };
}
```

MainResponses 1.- Diccionario de plantillas, métodos directos y llamada a métodos plantilla

```

public MainResponses()
{
    Register(new DictionaryRenderer(_responseTemplates));
}

public static IMessageActivity ConfusedResponses(ITurnContext turnContext, dynamic data)...
public static Dictionary<int, string> Responses()...

public static IMessageActivity BuildIntroCard(ITurnContext turnContext, dynamic data)
{
    var introCard = File.ReadAllText(MainStrings.INTRO_PATH);
    var card = AdaptiveCard.FromJson(introCard).Card;
    var attachment = new Attachment(AdaptiveCard.ContentType, content: card);

    var response = MessageFactory.Attachment(attachment, ssml: card.Speak, inputHint: InputHints.AcceptingInput);

    return response;
}
public static IMessageActivity BiblioAgradeciCard(ITurnContext turnContext, dynamic data)...
No Implementado

public class ResponseIds
{
    public const string Completed = "completed";
    public const string Confused = "confused";
    public const string Negacion_MostrarAnterior = "negacion_MostrarAnterior";

    public const string Intro = "intro";
    public const string BiblioAgradeci = "biblioAgradeci";
}
}

```

MainResponses 2.- Constructor y métodos plantilla

Según el gestor de respuestas del *Main*, *MainResponses*, plasmado en las imágenes

MainResponses 1.- Diccionario de plantillas, métodos directos y llamada a métodos plantilla y

MainResponses 2.- Constructor y métodos plantilla, un gestor de respuestas (clase derivada de *TemplateManager*) posee: un diccionario de plantillas, un constructor y métodos plantilla.

- *Diccionario de plantillas*. Gestiona las plantillas/respuestas contenidas por medio de un identificador de plantilla y pueden ser ejecutadas por *métodos directos* o *métodos plantilla*.
 Es posible crear plantillas para diferentes idiomas, incluyéndolas en grupos de plantillas, denominados mapas. Se pueden crear mapeos de cualquier idioma, pero este sistema resulta muy engorro de implementar. A raíz de esto, se ha implementado otro sistema que consiste en la Fijación del idioma en el middleware permitiendo mantener un solo mapeo, el mapeo por defecto (*default*).
 - *Métodos directos*: gestionan el mensaje de salida simplemente mediante la instrucción *MessageFactory.Text* que recoge la frase de salida del *archivo de recursos* correspondiente.
- *Constructor*. Simplemente crea y procesa el *diccionario de plantillas*.
- *Métodos plantilla*. Encargados de crear respuestas mediante *tarjetas adaptativas*, a partir de las rutas a los *archivos de notación* guardadas en los *archivos de recursos*.

Nota: Para que toda esta información siga una estructura ordenada, sea manipulable, de soporte a varios idiomas y pueda ser intercambiable son necesarios archivos de recursos y archivos de notación.

3.4.1. Archivos de recursos y archivos de notación

Los **archivos de recursos** (.resx) guardan texto corto y enlaces a *archivos de notación* (Archivo de recursos 1.- Archivo de recursos del MainDialog, MainStrings.resx).

Los **archivos de notación** (.json) redactan toda la información sobre una *tarjeta adaptativa* como la *Tarjeta de la Bibliografía* (Archivo de notación 1.- Fragmento de la tarjeta de la Bibliografía).

	Nombre	Valor
	BIBLIOAGRADECIMIENTO_PATH	.\Dialogs\Main\Resources\BiblioAgradecimiento.json
	CANCELLED	Ok, let's start over.
	COMPLETED	What else can I help you with?
	CONFUSED0	I'm sorry, I'm not able to help with that.
	CONFUSED1	Perdon, no te he entendido.
	CONFUSED2	No he entendido tu petición.
	CONFUSEDMAX	No te entiendo. _Sugerencia:_ Escribe _tarjeta de introducción_ para acceder
	DESPUES_INDICE	Búsqueda: **capítulo nº apartado** _Ejemplo:_ ***capítulo 2 programación
	ENLACE1_TEXTO	tutorialsPoint. Referencia C
	ENLACE1_VALOR	https://www.tutorialspoint.com/cprogramming
	ENLACE2_TEXTO	cppreference. Referencia C
	ENLACE2_VALOR	https://es.cppreference.com/w/c
	INTRO_PATH	.\Dialogs\Main\Resources\Intro.json
	NEGACION_MOSTRARANTERIOR	No se ha consultado ningún apartado anteriormente. _Empieza ahora!
*	PROBLEMAS_PATH	.\Dialogs\PROBLEMAS

Archivo de recursos 1.- Archivo de recursos del MainDialog, MainStrings.resx

```

1 ? { "type": "AdaptiveCard",
2   "body": [
3     {
4       "type": "ColumnSet",
5       "horizontalAlignment": "Left",
6       "columns": [
7         {
8           "type": "Column",
9           "horizontalAlignment": "Left",
10          "items": [
11            {
12              "type": "TextBlock",
13              "horizontalAlignment": "Center",
14              "spacing": "Small",
15              "size": "ExtraLarge",
16              "color": "Dark",
17              "text": "__B__",
18              "isSubtle": true
19            },
20            {
21              "type": "TextBlock",
22              "horizontalAlignment": "Center",
23              "spacing": "Small",
24              "size": "ExtraLarge",
25              "color": "Dark",
26              "text": "__I__",
27              "isSubtle": true
28            },
29            {
30              "type": "TextBlock",
31              "horizontalAlignment": "Center",
32              "spacing": "Small",
33              "size": "ExtraLarge",
34              "color": "Dark",
35              "text": "__B__",
36              "isSubtle": true
37            }

```

Archivo de notación 1.- Fragmento de la tarjeta de la Bibliografía

Un *archivo de notación* solo puede ser lanzado por *métodos plantilla* porque necesita código extra para interpretar *tarjetas adaptativas*, mientras que un punto del *archivo de recursos* puede ser lanzado bien por *métodos plantilla* o bien por *métodos directos*.

Nota: Tanto los archivos de recursos como los archivos de notación pueden utilizar marcaje MARKDOWN.

Fijación del idioma en el middleware

Como se ha comentado anteriormente, ambos archivos pueden ser lanzado en idiomas diferentes y, de hecho, el Bot posee las bases para la implementación de cualquier idioma. El sistema elegido es la **fijación del idioma en el middleware**.

Este sistema, en primer lugar, fija el idioma en el *middleware*, *SetLocaleMiddleware*, cuyo método *OnTurnAsync* es el encargado como muestra la imagen SetLocaleMiddleware 1.- Fijación del idioma según la localización. En segundo lugar, hay que recoger el texto en el idioma correspondiente, de lo cual, se encarga el *diseñador* (clase) del *archivo de recursos* como muestra la imagen MainStrings.Designer.cs 1.- diseñador del archivo de recursos del MainDialog.



```

public class SetLocaleMiddleware : IMiddleware
{
    private readonly string _defaultLocale;

    public SetLocaleMiddleware(string defaultDefaultLocale)
    {
        _defaultLocale = defaultDefaultLocale;
    }

    public async Task OnTurnAsync(ITurnContext context, NextDelegate next, CancellationToken cancellationToken = default(CancellationToken))
    {
        var localeWebApp = context.Activity.Locale;
        var cultureInfo = string.IsNullOrEmpty(localeWebApp) ? new CultureInfo(_defaultLocale) : new CultureInfo(localeWebApp);

        CultureInfo.CurrentCulture = CultureInfo.CurrentUICulture = cultureInfo;

        await next(cancellationToken).ConfigureAwait(false);
    }
}

```

SetLocaleMiddleware 1.- Fijación del idioma según la localización

Según muestra la imagen SetLocaleMiddleware 1.- Fijación del idioma según la localización:

En cada turno de conversación (*OnTurnAsync*) se fija el idioma del Bot (*CurrentUICulture*) dependiendo en qué parte del mundo este el Usuario conectado al canal, *Activity.Locale*. Por defecto, el idioma (*CurrentUICulture*) se fija a “es-ES”.

```

/// <summary>
/// Reemplaza la propiedad CurrentUICulture del subproceso actual para todas las
/// búsquedas de recursos mediante esta clase de recurso fuertemente tipado.
/// </summary>
[global::System.ComponentModel.EditorBrowsableAttribute(global::System.ComponentModel.EditorBrowsableState.Advanced)]
public static global::System.Globalization.CultureInfo Culture {
    get {
        return resourceCulture;
    }
    set {
        resourceCulture = value;
    }
}

/// <summary>
/// Busca una cadena traducida similar a .\Dialogs>Main\Resources\BiblioAgradeci.json.
/// </summary>
public static string BIBLIOAGRADECI_PATH {
    get {
        return ResourceManager.GetString("BIBLIOAGRADECI_PATH", resourceCulture);
    }
}

```

MainStrings.Designer.cs 1.- diseñador del archivo de recursos del MainDialog

Según muestra la imagen MainStrings.Designer.cs 1.- diseñador del archivo de recursos del MainDialog:

El *diseñador* puede gestionar el idioma de la respuesta (*CurrentUICulture* fijado en el *middleware*) accediendo al *archivo de recursos* con extensión correspondiente.



Por ejemplo, para acceder a la *Tarjeta de la Bibliografía (BiblioAgradeci.json)* con idioma “es-ES”; el *diseñador* accederá, primero, al *archivo de recursos* del *MainDialog* con extensión en español, “MainStrings.es.resx”, que contendrá la ruta a “BiblioAgradeci.es.json”.

La *fijación del idioma en el middleware* frente a la creación de grupos de plantillas (mapas) para cada idioma, hace que se escriba muchísimo menos código. No es lo mismo crear un mapa para cada idioma (implicando la creación del mapa, la creación de los *métodos plantilla* correspondientes y la ampliación del *archivo de recursos*) que usar el *diseñador* del *archivo de recursos* (implicando la creación de un *archivo de recursos* para cada idioma).

3.4.2. Respuestas simples

Una *respuesta simple* o actividad de salida simple se puede lanzar mediante la función *ReplyWith* de un gestor de respuestas (*_responder/TemplateManager*) determinado.

```
await _responder.ReplyWith(innerDc.Context, MainResponses.ResponseIds.BiblioAgradeci);
```

Código 3.- Mostrar la tarjeta de la Bibliografía

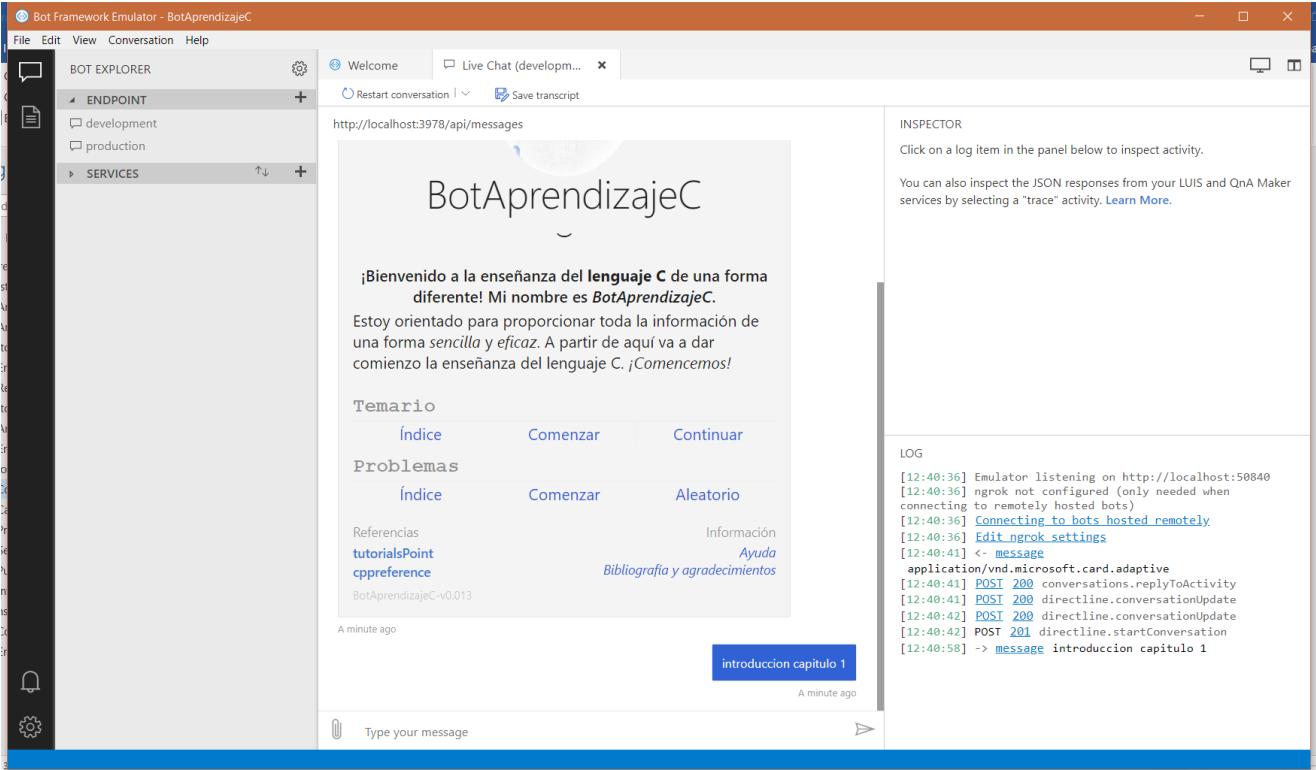
Como muestra la imagen Código 3.- Mostrar la tarjeta de la Bibliografía, al utilizar la función *ReplyWith* de un gestor de respuestas (*_responder*), pasando el *contexto del turno* y el identificador de plantilla, se puede lanzar una respuesta/plantilla como mensaje de salida, por ejemplo, la tarjeta de la *Bibliografía*.

3.5. Recorrido de un punto del APRENDIZAJE DE C

Para tratar el funcionamiento sobre la ejecución de un punto del APRENDIZAJE DE C se va a coger el punto -Introducción al Capítulo 1- del TEMARIO.

Nota: Hay que recalcar que los demás puntos del APRENDIZAJE DE C, tanto del TEMARIO como de los PROBLEMAS, siguen un comportamiento similar.

3.5.1. Comienzo



Bot Framework Emulator 3.- Parte tarjeta de Introducción y mensaje de entrada “introduccion capitulo 1”

Para comenzar el Usuario por ejemplo escribe “introduccion capitulo 1” (Bot Framework Emulator 3.- Parte tarjeta de Introducción y mensaje de entrada “introduccion capitulo 1”) dando comienzo la actividad de entrada. Tras haber ejecutado todas las funciones *OnTurnAsync* del *middleware*, se pasa el funcionamiento al método *OnTurnAsync* del Punto de entrada de mensajería.

Ahora, da comienzo el *MainDialog* por su clase herencia *RouterDialog*; más concretamente por *OnContinueDialogAsync* que inicia el método *ReconocimientoExpresionesAsync* del *MainDialog* (imágenes *RouterDialog* 2.- Método CONTINUADOR Parte 1 y *RouterDialog* 3.- Método CONTINUADOR Parte 2).

Una vez reconocida la intención del texto de la actividad de entrada, *Activity.Text*, se ejecuta el método *BeginDialogAsync* (del contexto del diálogo, *DialogContext*) con identificador “Cap01IntroduccionDialog” (imágenes *MainDialog* 7.- Método ReconocimientoExpresionesAsync Parte 1 y *MainDialog* 10.- Método ReconocimientoExpresionesAsync Parte 4) dando comienzo dicho apartado del TEMARIO.

Importante: cualquier punto del APRENDIZAJE DE C es un diálogo en cascada que su vez, hereda de dos diálogos encargados de controlar las interrupciones (*AprendizajeCDialog* y *InterruptableDialog* respectivamente) y, por último, del diálogo de componentes.

3.5.2. Campos y constructor del diálogo en cascada

```
public class Cap01IntroduccionDialog : AprendizajeCDialog
{
    private static Cap01IntroduccionResponses _responder = new Cap01IntroduccionResponses();
    private static SharedTEMARIOResponses _sharedResponder = new SharedTEMARIOResponses();

    private IStatePropertyAccessor<UserPropertiesState> _accessor;

    public Cap01IntroduccionDialog(BotServices botServices, IStatePropertyAccessor<UserPropertiesState> accessor, IBotTelemetryClient
        : base(botServices, nameof(Cap01IntroduccionDialog))
    {
        _accessor = accessor;
        InitialDialogId = nameof(Cap01IntroduccionDialog);

        // Se debe fijar la telemetría tanto en el dialogo de Componentes como en el dialogo de Cascada.
        TelemetryClient = telemetryClient;
        AddDialog(new WaterfallDialog(InitialDialogId, new WaterfallStep[] { ShowText, Finish, FinishOther })) { TelemetryClient = telemetryClient };
        AddDialog(new ChoicePrompt(DialogIds.Choice));

        //Dialogo de reemplazo
        AddDialog(new Cap0201IntroduccionDialog(botServices, accessor, telemetryClient));
    }
}
```

Cap01IntroduccionDialog 1.- Gestores de respuestas y constructor

Según muestra la imagen Cap01IntroduccionDialog 1.- Gestores de respuestas y constructor.

- Se inicializan los gestores de respuestas para: el apartado del TEMARIO y el compartido para todo el TEMARIO.
- Se ejecuta el constructor, que acepta los servicios (modelos **Luis**) y el descriptor de acceso a las propiedades del Usuario. Los servicios son pasados a las clases herencia que los utilizarán para controlar las interrupciones.
 - El descriptor de acceso es almacenado en el campo “_accessor”.
 - La propiedad *InitialDialogId*, heredada de *ComponentDialog*, hace que el diálogo actual *Cap01IntroduccionDialog*, como *diálogo de componentes*, pueda iniciar por otro diálogo que lleve el mismo identificador “*Cap01IntroduccionDialog*”.
 - La telemetría no se utiliza, pero el Bot está construido para que en un momento dado se puede utilizar.
 - Se añade un *diálogo en cascada* con tres pasos: *ShowText*, *Finish*, *FinishOther* y se le da el mismo nombre del *InitialDialogId* haciendo que el diálogo actual, *Cap01IntroduccionDialog*, comience por el *diálogo en cascada*, más concretamente, por el primer paso de la cascada.
 - Se añade un *diálogo de petición* del tipo elección con identificador *DialogIds.Choice*.
 - Se añade un diálogo de reemplazo que servirá de enlace al siguiente apartado del TEMARIO.

3.5.3. Primer paso de la cascada

```
public async Task<DialogTurnResult> ShowText(WaterfallStepContext sc, CancellationToken cancellationToken)
{
    //Almacena el ultimo apartado consultado
    var _state = await _accessor.GetAsync(sc.Context, () => new UserPropertiesState());
    _state.UltimoApartado = sc.ActiveDialog.Id;

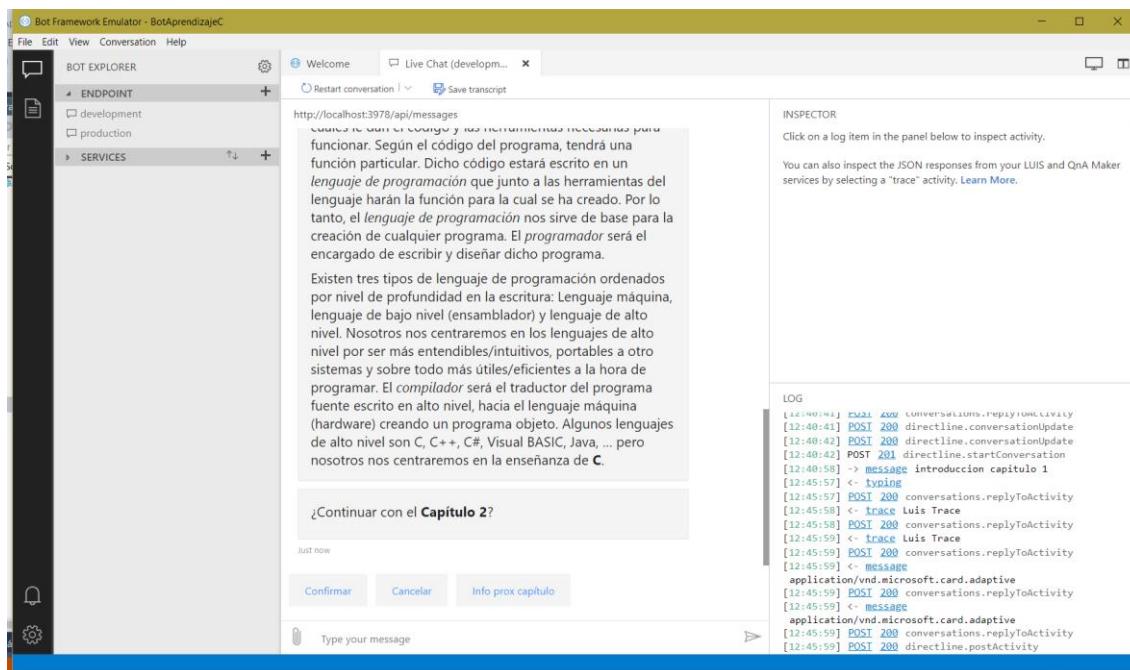
    //Info apartado capitulo
    await _responder.ReplyWith(sc.Context, Cap01IntroduccionResponses.ResponseIds.Cap01Introduccion);

    return await sc.PromptAsync(DialogIds.Choice, new PromptOptions()
    {
        Prompt = await _sharedResponder.RenderTemplate(sc.Context,
            sc.Context.Activity.Locale, SharedTEMARIOResponses.ResponseIds.SigCap,
            new {
                NumSigCap = SharedTEMARIOFunctions.GetNum(nameof(Cap01IntroduccionDialog))
            }),
        Choices = ChoiceFactory.ToChoices(new List<string> { SharedStrings.CONFIRMAR_TEXTO, SharedStrings.CANCELAR_TEXTO,
            SharedStrings.INDSIGCAP_TEXTO })
    });
}
```

Cap01IntroduccionDialog 2.- Primer paso de la cascada

La imagen Cap01IntroduccionDialog 2.- Primer paso de la cascada dice:

- El primer paso de la cascada de cualquier punto del APRENDIZAJE DE C siempre guarda el último apartado consultado. Gracias al descriptor de acceso a las propiedades del Usuario, `_accessor`, se pudo obtener (`GetAsync`) el estado de las propiedades y entonces actuar sobre ellas. Se almacenará en la propiedad `UltimoApartado` la `Id` del diálogo activo, en este caso, “Cap01IntroduccionDialog”.
- Ahora se envía un mensaje de salida (desde un gestor de respuestas) mostrando la información, en este caso, sobre la introducción al capítulo 1.
- Por último, se activa el *peticionador o respuesta elaborada*, `PromptAsync`, que, mediante un tipo de diálogo y unas opciones, envía un mensaje de salida compuesto por una pregunta (texto) y alternativas a elegir (Botones de acción) por el Usuario.



Bot Framework Emulator 4.- Parte de la información sobre la Introducción al capítulo 1 y mensaje de salida más elecciones

Tras estos pasos se mostrará en el emulador lo mismo que en la imagen Bot Framework Emulator 4.- Parte de la información sobre la Introducción al capítulo 1 y mensaje de salida más elecciones.

Al finalizar el primer paso de la cascada se devolverá el control al *MainDialog* terminando el método *ReconocimientoExpresionesAsync* (*MainDialog* 10.- Método *ReconocimientoExpresionesAsync* Parte 4) y pasando el control al método *OnContinueDialogAsync* del *RouterDialog* (*RouterDialog* 3.- Método *CONTINUADOR* Parte 2) que terminará el turno del Bot, *EndOfTurn*.

Nota: Una vez finalizado el turno del Bot, se lanzará de nuevo el middleware y se autoguardarán los estados de la Conversación y del Usuario (AutoSaveStateMiddleware). Como recordatorio el sistema de guardado elegido fue el autoguardado en el middleware en contraposición del guardado manual.

3.5.4. Respuestas elaboradas

Una *respuesta elaborada* o actividad de salida elaborada se realiza mediante un método *peticionador*, *PromptAsync*, que acepta el tipo de *diálogo de petición* y las opciones que llevará el diálogo (Cap01IntroduccionDialog 2.- Primer paso de la cascada).

- Los tipos de *diálogo de petición* pueden ser de texto, elección, confirmación, número, etc. En este caso se añadió al constructor un *diálogo de petición* de elección, *ChoicePrompt*. Cada tipo de *diálogo de petición* posee unos validadores, *PromptValidator*, internos que sirven de seguridad ante respuestas erróneas, por ejemplo, al mostrar unas opciones no es posible escribir otra distinta. En caso, de que la respuesta del Usuario no sea válida, se volverá a formular el *peticionador* hasta que lo sea. Es posible crear validadores personalizados.
- Las opciones del *peticionador*, *PromptOptions*, son:
 - *Prompt*. Almacena el mensaje de la petición mediante el método *RenderTemplate* que devuelve una actividad a partir de una plantilla del gestor de respuestas.
 - *RetryPrompt*. Es posible mostrar una respuesta alternativa, cuando la respuesta al *peticionador* no es validada, del mismo modo que para la opción *Prompt*.
 - *Choices*. Almacena las elecciones disponibles mediante un almacén de elecciones, *ChoiceFactory.ToChoices*, que convierte una lista de cadenas en botones de acción.

return enviará un estado “en espera”, *DialogTurnStatus.Waiting*, al finalizar el primer paso de la cascada.

3.5.5. Control de interrupciones

Al pulsar un botón o escribir se desata la actividad/mensaje de entrada dando paso a los métodos *OnTurnAsync* del *middleware*. El último componente del *middleware* pasa el control al Punto de entrada de mensajería. Como ahora hay un diálogo activo se continuará la conversación abriendo el primer diálogo en la pila de diálogos, es decir, el *MainDialog* y por consiguiente su diálogo heredado *RouterDialog*. El método *OnContinueDialogAsync* será el encargado de actuar (ya que se está continuando un diálogo) como se muestra en *RouterDialog*

2.- Método CONTINUADOR Parte 1 y desembocará en la continuación (*ContinueDialogAsync*) del segundo diálogo activo en la pila “Cap01IntroduccionDialog”.

Advertencia: Se podría esperar que la continuación fuera por el segundo paso de la cascada, pero no hay que olvidar que, cualquier punto del Aprendizaje de C posee diálogos heredados (AprendizajeCDialog, InterruptableDialog y ComponentDialog respectivamente).

Entonces, al ser *ContinueDialogAsync* el método utilizado y no *BeginDialogAsync*, se llama al método *OnContinueDialogAsync* anulado de *ComponentDialog* más cercano, es decir, el de *InterruptableDialog* (sino hubiese sido anulado se continuaría por el siguiente paso de la cascada) como se muestra *InterruptableDialog*

- 1.-Constructor, *OnContinueDialogAsync* anulado y *OnDialog INTERRUPTIONAsync* creado abstracto.

El método CONTINUADOR llama a un método creado que controlar las interrupciones, *OnDialog INTERRUPTIONAsync*.

```
/// <summary> Manejar las interrupciones del dialogo activo. Clase derivada de C ...
public abstract class InterruptableDialog : ComponentDialog
{
    /// <summary> Constructor mediador entre AprendizajeCDialog y ComponentDialog.
    public InterruptableDialog(string dialogId)
        : base(dialogId)
    {
    }

    /// <summary> CONTINUADOR del InterruptableDialog invalidado del ComponentDialog ...
    protected override async Task<DialogTurnResult> OnContinueDialogAsync(DialogContext inner2Dc, CancellationToken cancellationToken)
    {
        var status = await OnDialog INTERRUPTIONAsync(inner2Dc, cancellationToken);

        if (status == InterruptionStatus.Waiting)
            // A la espera de un respuesta del 'CancelDialog'
            return EndOfTurn;
        }
        [Interruptido. No utilizado]
        return await base.OnContinueDialogAsync(inner2Dc, cancellationToken);
    }

    /// <summary> Metodo abstracto ha invalidar en la clase derivada.
    protected abstract Task<InterruptionStatus> OnDialog INTERRUPTIONAsync(DialogContext inner2Dc, CancellationToken cancellationToken);
}
```

InterruptableDialog 1.-Constructor, *OnContinueDialogAsync* anulado y *OnDialog INTERRUPTIONAsync* creado abstracto

Como este método es abstracto será buscado en clases derivadas de esta, concretamente en *AprendizajeCDialog* como muestra *AprendizajeCDialog* 2.- *OnDialog INTERRUPTIONAsync* anulado.

```
/// <summary> Clase herencia de todos los dialogos del "Aprendizaje de C".
public class AprendizajeCDialog : InterruptableDialog
{
    private readonly BotServices _services;

    public AprendizajeCDialog(BotServices botServices, string dialogId)
        : base(dialogId)
    {
        _services = botServices;
        AddDialog(new CancelDialog());
    }
}
```

AprendizajeCDialog 1.- Constructor

Esta clase, AprendizajeCDialog 1.- Constructor, recibe los servicios ([LUIS](#)) y añade el diálogo de cancelación, *CancelDialog*, que será usado más adelante.

```
/// <summary> Metodo que analiza las posibles interrupciones
protected override async Task<InterruptedException> OnDialogInterruptionAsync(DialogContext inner2Dc, CancellationToken cancellationToken)
{
    var generalRecognizer = _services.GeneralRecognizer;

    if (generalRecognizer == null)
        throw new Exception("El modelo LUIS especificado no pudo ser encontrado en la configuracion 'BotServices'.");
    else
    {
        var generalResult = await generalRecognizer.RecognizeAsync<General>(inner2Dc.Context, true, cancellationToken);
        var generalIntent = generalResult.TopIntent().intent;

        // La interrupcion solo se activa si el nivel de confianza es alto
        if (generalResult.TopIntent().score > 0.5)
        {
            Añade el resultado de Luis (intentos y entidades) para mayor procesamiento en dialogos derivados
            switch (generalIntent)
            {
                case General.Intent.Cancel:
                    return await OnCancel(inner2Dc);
                case General.Intent.Help:
                    return await OnHelp(inner2Dc);
            }
        }
    }
    return InterruptionStatus.NoAction;
}
```

AprendizajeCDialog 2.- OnDialogInterruptionAsync anulado

Según la imagen AprendizajeCDialog 2.- OnDialogInterruptionAsync anulado, mediante el modelo **General** se comprueba la intención del mensaje de entrada, si la intención es *Cancel* o *Help* se lanzará el método *OnCancel* u *OnHelp*. Si no, se retornará que no se ha interrumpido el diálogo, *InterruptedException.NoAction*.

```

protected virtual async Task<InterruptionStatus> OnCancel(DialogContext inner2Dc)
{
    // No vuelve a comenzar el dialogo cancelar, si ya se esta en el
    if (inner2Dc.ActiveDialog.Id != nameof(CancelDialog))
    {
        await inner2Dc.BeginDialogAsync(nameof(CancelDialog));

        // Señala que el dialogo esta esperando a la respuesta del usuario
        return InterruptionStatus.Waiting;
    }

    // Sino, continuar
    return InterruptionStatus.NoAction;
}

protected virtual async Task<InterruptionStatus> OnHelp(DialogContext inner2Dc)
{
    // No vuelve a comenzar el dialogo cancelar, si ya se esta en el
    if (inner2Dc.ActiveDialog.Id != nameof(CancelDialog))
    {
        await inner2Dc.BeginDialogAsync(nameof(CancelDialog),
            new { ID1 = CancelResponses.ResponseIds.IniHelpCancelID,
                  ID2 = CancelResponses.ResponseIds.EndHelpCancelID });

        // Señala que el dialogo esta esperando a la respuesta del usuario
        return InterruptionStatus.Waiting;
    }

    // Sino, continuar
    return InterruptionStatus.NoAction;
}
}

```

AprendizajeCDialog 3.- OnCancel y OnHelp

Según la imagen AprendizajeCDialog 3.- OnCancel y OnHelp:

- *OnCancel*. Comienza el diálogo de cancelación, *CancelDialog*, (en caso de no estar ya activo) lo coloca en la pila de diálogos y devuelve un estado del diálogo “en espera”, *InterruptionStatus.Waiting*. Si el *CancelDialog* ya está activo se devuelve un estado *InterruptionStatus.NoAction*.
- *OnHelp*. Comienza el diálogo de cancelación, *CancelDialog*, (en caso de no estar ya activo) pasando información adicional que lo modificará, lo coloca en la pila de diálogos y devuelve un estado del diálogo “en espera”, *InterruptionStatus.Waiting*. Si el *CancelDialog* ya está activo se devuelve un estado *InterruptionStatus.NoAction*.

Una vez terminado el método *OnDialogInterruptionAsync* se comprobará el estado devuelto. Si el estado está “en espera” (es decir, se ha comenzado y ejecutado el primer paso de la cascada del diálogo de cancelación, *CancelDialog*) el turno del Bot finaliza, *EndOfTurn*. Si no es así se inicia el método *OnContinueDialogAsync* del diálogo base (*ComponentDialog*), que inicia el segundo paso de la cascada del diálogo Cap01IntroduccionDialog.

3.5.6. Segundo paso de la cascada

```
public async Task<DialogTurnResult> Finish(WaterfallStepContext sc, CancellationToken cancellationToken)
{
    var Option = sc.Context.Activity.Text.ToLower();
    if(Option == SharedStrings.CONFIRMAR_TEXTO.ToLower())
    {
        return await sc.ReplaceDialogAsync(nameof(Cap0201IntroduccionDialog));
    }
    else if(Option == SharedStrings.INDSIGCAP_TEXTO.ToLower())
    {
        return await sc.PromptAsync(DialogIds.Choice, new PromptOptions()
        {
            Prompt = await _sharedResponder.RenderTemplate(sc.Context, sc.Context.Activity.Locale,
                SharedTEMARIOResponses.ResponseIds.IndiceSigCap,
                new {
                    NumSigCap = SharedTEMARIOFunctions.GetNum(nameof(Cap01IntroduccionDialog)),
                    Cap = SharedStrings.CAP02,
                    SigCap = SharedStrings.SIG_CAP
                }),
            Choices = ChoiceFactory.ToChoices(new List<string> { SharedStrings.CONFIRMAR_TEXTO, SharedStrings.CANCELAR_TEXTO });
        });
    }
    else
    {
        return await sc.EndDialogAsync();
    }
}
```

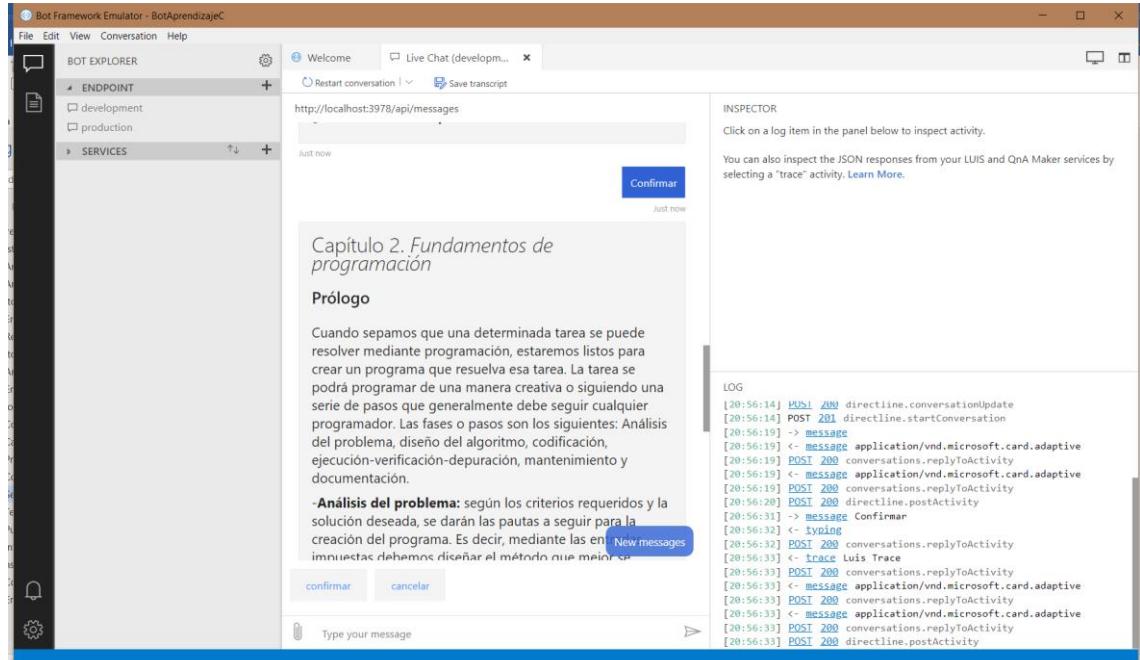
Cap01IntroduccionDialog 3.- Segundo paso de la cascada

Tanto si el Usuario pulsa un botón como si escribe, el método *OnContinueDialogAsync* de *RouterDialog* actuará metiendo la información en el texto de la actividad, *Activity.Text* (RouterDialog 2.- Método CONTINUADOR Parte 1).

Teniendo lo anterior en mente el segundo paso de la cascada, Cap01IntroduccionDialog 3.- Segundo paso de la cascada, comprobará la opción almacenada en *Activity.Text* y actuará según la opción elegida:

- Confirmar. El método *ReplaceDialogAsync* sustituye el diálogo activo, *Cap01IntroduccionDialog*, (segundo diálogo activo en la pila) por otro, en este caso, el siguiente punto del APRENDIZAJE DE C como se muestra en Bot Framework Emulator 5.-

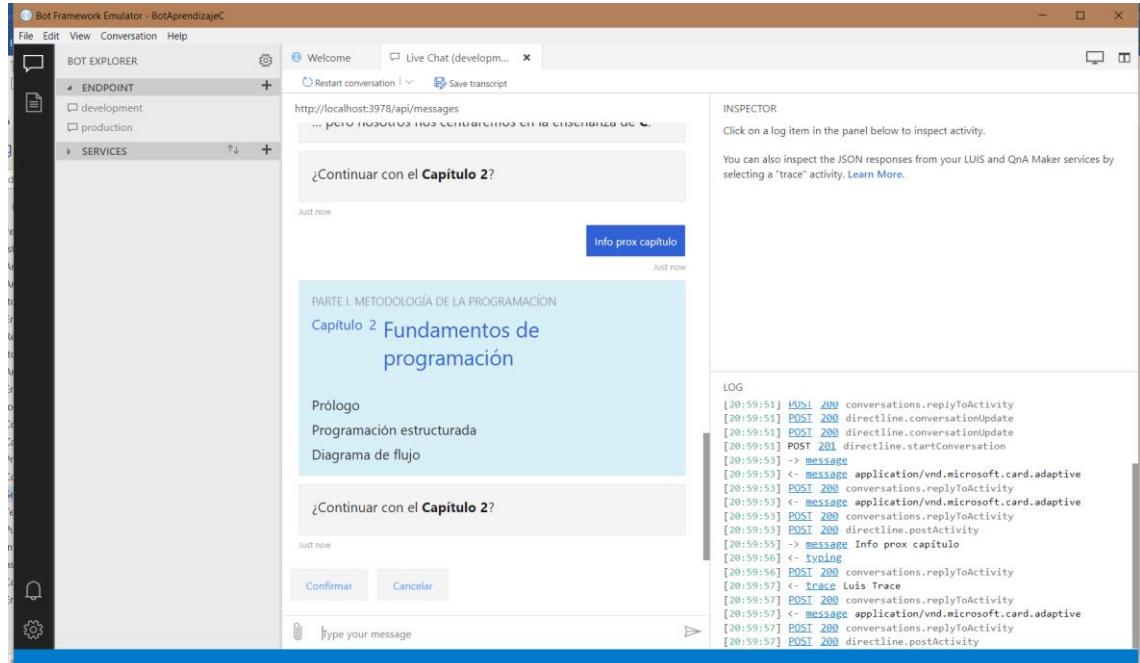
“Confirmar” y reemplazo del diálogo por el siguiente punto del APRENDIZAJE DE C.



Bot Framework Emulator 5.- “Confirmar” y reemplazo del diálogo por el siguiente punto del APRENDIZAJE DE C

- Índice al siguiente capítulo. Como el capítulo 1 solo posee un punto/apartado, se da la opción de visualizar el índice del siguiente capítulo antes de comenzarlo. De esta manera, se lanzará el *peticionario* de nuevo mostrando el índice del capítulo más la pregunta, ahora, con las opciones “Confirmar” y “Cancelar” como se muestra en Bot Framework Emulator 6.- “Info prox capítulo” y muestra del índice al siguiente capítulo

más un nuevo mensaje de elección.



Bot Framework Emulator 6.- "Info prox capítulo" y muestra del índice al siguiente capítulo más un nuevo mensaje de elección

3.5.7. Tercer paso de la cascada

```
public async Task<DialogTurnResult> FinishOther(WaterfallStepContext sc, CancellationToken cancellationToken)
{
    var ConfirmNextDialog = sc.Context.Activity.Text.ToLower();
    if (ConfirmNextDialog == SharedStrings.CONFIRMAR_TEXTO.ToLower())
    {
        return await sc.ReplaceDialogAsync(nameof(Cap0201IntroduccionDialog));
    }
    else
    {
        return await sc.EndDialogAsync();
    }
}
```

Cap01IntroduccionDialog 4.- Tercer paso de la cascada

Una vez el Usuario pulse un botón o escriba (obviando el Middleware, el Punto de entrada de mensajería, el Encauzamiento de los mensajes de entrada y el Control de interrupciones), según muestra la imagen Cap01IntroduccionDialog 4.- Tercer paso de la cascada, se comprobará si se ha confirmado y entonces se reemplazará el diálogo activo, Cap01IntroduccionDialog, por el siguiente punto del APRENDIZAJE DE C.

3.5.8. CancelDialog

El diálogo de cancelación, *CancelDialog*, se puede desencadenar en cada turno de conversación mientras se esté consultando un punto del APRENDIZAJE DE C.

El acceso al diálogo se realiza por el Control de interrupciones siguiendo la serie de métodos *OnContinueDialogAsync* y *OnDialog INTERRUPTIONAsync*, y, por último, *OnCancel* u *OnHelp* mostrados en AprendizajeCDialog 3.- OnCancel y OnHelp.

```
/// <summary> Da la posibilidad de cancelar (los dialogos activos).</summary>
public class CancelDialog : ComponentDialog
{
    private static CancelResponses _responder = new CancelResponses();

    public CancelDialog()
        : base(nameof(CancelDialog))
    {
        InitialDialogId = nameof(CancelDialog);

        var cancel = new WaterfallStep[] { AskToCancel, FinishCancelDialog };

        AddDialog(new WaterfallDialog(InitialDialogId, cancel));
        AddDialog(new ChoicePrompt(DialogIds.CancelPrompt));
    }

    private async Task<DialogTurnResult> AskToCancel(WaterfallStepContext sc, CancellationToken cancellationToken)
    {
        dynamic options = sc.Options;
        if (options != null) // Cancelacion personalizada
            return await sc.PromptAsync(DialogIds.CancelPrompt, new PromptOptions()
            {
                Prompt = await _responder.RenderTemplate(sc.Context, sc.Context.Activity.Locale, options.ID1.ToString()),
                Choices = ChoiceFactory.ToChoices(new List<string> { CancelStrings.YES, CancelStrings.NO })
            });
        else //Cancelacion por defecto
            return await sc.PromptAsync(DialogIds.CancelPrompt, new PromptOptions()
            {
                Prompt = await _responder.RenderTemplate(sc.Context, sc.Context.Activity.Locale,
                    CancelResponses.ResponseIds.CancelPrompt),
                Choices = ChoiceFactory.ToChoices(new List<string> { CancelStrings.YES, CancelStrings.NO })
            });
    }
}
```

CancelDialog 1.- Constructor y primer paso de la cascada

Según muestra la imagen CancelDialog 1.- Constructor y primer paso de la cascada:

CancelDialog deriva del *diálogo de componentes*, *ComponentDialog*, y se compone de un *diálogo en cascada* y un *diálogo de petición* de elección. El diálogo comenzará por el *diálogo en cascada*, más concretamente, por el primer paso de la cascada, porque posee el mismo identificador que la propiedad *InitialDialogId* (de *ComponentDialog*).

El primer paso de la cascada comprueba si se han pasado opciones al comenzar el diálogo (*BeginDialogAsync(dialogid, Options, cancellationToken)*) y entonces, se lanza o bien la cancelación personalizada o bien la cancelación por defecto. Cada una lanza un *peticionario* de elección (Si, No) con un mensaje distinto.

```

private async Task<DialogTurnResult> FinishCancelDialog(WaterfallStepContext sc, CancellationToken cancellationToken)
{
    return await sc.EndDialogAsync(new { YesNo = sc.Result, options = sc.Options });
}

/// <summary> Llamado cuando se finaliza el dialogo activo, EndDialogAsync. Meto ...
protected override async Task<DialogTurnResult> EndComponentAsync(DialogContext outerDc, dynamic result, CancellationToken
{
    var doCancel = result.YesNo.Value.ToLower();
    if (doCancel == CancelStrings.YES.ToLower())
    {
        if (result.options != null) // Cancelacion personalizada
        {
            await _responder.ReplyWith(outerDc.Context, result.options.ID2.ToString());
            return await outerDc.CancelAllDialogsAsync();
        }
        else // Cancelacion por defecto
        {
            await _responder.ReplyWith(outerDc.Context, CancelResponses.ResponseIds.CancelConfirmedMessage);
            return await outerDc.CancelAllDialogsAsync();
        }
    }
    else
    {
        await _responder.ReplyWith(outerDc.Context, CancelResponses.ResponseIds.CancelDeniedMessage);
        //Solo finaliza el CancelDialog
        return await outerDc.EndDialogAsync();
    }
}

private class DialogIds
{
    public const string CancelPrompt = "cancelPrompt";
}

```

CancelDialog 2.- Segundo paso de la cascada y EndComponentAsync

Una vez recibida la respuesta del Usuario (mensaje/actividad de entrada) (obviando el Middleware, el Punto de entrada de mensajería y el Encauzamiento de los mensajes de entrada), el segundo paso de la cascada llama al método *EndDialogAsync* (pasándole una variable anónima con el mensaje de recibido, *Result*, y las posibles opciones, *Options*).

EndDialogAsync comienza automáticamente *EndComponentAsync* (anulado del *ComponentDialog*).

- Si se ha elegido “No” se mostrará un mensaje de denegación de la cancelación y finalizará el *CancelDialog* (*EndDialogAsync*).
- Si se ha elegido “Si” y se han pasado opciones (*Options*) se enviará un mensaje personalizado de cancelación. Entonces, se cancelarán todos los diálogos activos en la pila excepto, los diálogos añadidos al diálogo fijo (*DialogSet*), en este caso, el *MainDialog*.
- Si se ha elegido “Si” y no se han pasado opciones (*Options*) se enviará un mensaje normal de cancelación. Entonces, se cancelarán todos los diálogos activos en la pila excepto, los diálogos añadidos al diálogo fijo (*DialogSet*), en este caso, el *MainDialog*.

3.6.LUIS App. Administración y desarrollo

Importante: Toda la información mostrada en este punto, *LUIS* App. Administración y desarrollo , muestra el funcionamiento de la aplicación *LUIS* referenciada en las Herramientas de construcción, concretamente, en la sección Servicios Cognitivos.

Luis, Language understanding o Reconocimiento del lenguaje fue diseñado con el objetivo de simplificar la metodología utilizada para reconocer la información importante de cada conversación.

Luis interpreta la conversación del Usuario (mensaje de entrada) y devuelve su objetivo o intención (*intent*). Adicionalmente, puede devolver la información de valor o entidades (*entities*) incrustadas en dicha conversación.

- Ejemplo - **Intención** (*intent*). Para el mensaje de entrada “de que se compone un ordenador” se devolvería una intención (*intent*) “ComposicionOrdenador” que el código utilizaría para desencadenar una información concreta.
- Ejemplo - **Intención con entidades** (*entities*). Si ahora también se usasen entidades, *entities*, que hicieran referencia a la composición de diferentes objetos; para los mensajes de entrada “de que se compone un ordenador” o “de que se compone un coche”, se devolvería una intención (*intent*) “Composicionde” y unas entidades (*entities*) “Ordenador” o “Coche” permitiendo un desencadenamiento de la información más preciso y simplificado.

Nota: Toda la información que contiene un modelo Luis debe ser implementada manualmente. La información mostrada en los ejemplos tiene que implementarse también, es decir, no existe un modelo estándar que acepte y devuelva lo explicado.

Es uso de intenciones y entidades hace que la creación de un modelo **Luis** sea más compleja que si solo poseyese intenciones. No es tanto el incluir entidades en sí, sino cuales y como se van a incluir y si son suficientes para desencadenar la información.

En general el uso de entidades simplificaría el modelo, es decir, habría que redactar menos. En contraposición, el código necesario para desencadenar la información quedaría más complejo.

3.6.1. Aplicación Luis

Mis aplicaciones

+ Crear aplicación		Importar nueva aplicación	Buscar aplicaciones...
<input type="checkbox"/> Nombre	Referencia cultural	Fecha de creación	Visitas de puntos de conexión
ProblemasC (V 0.1)	es-es	3/21/19	298
TemarioC (V 0.1)	es-es	1/29/19	644
General (V 0.1)	es-es	1/24/19	1374
Dispatch (V 0.1)	es-es	1/24/19	1284

Luis App 1.- Mis aplicaciones/modelos

Según muestra la imagen Luis App 1.- Mis aplicaciones/modelos:

Nada más acceder a la aplicación **Luis** en la Web (luis.ai), se visualizan las aplicaciones creadas anteriormente. Además, se da la posibilidad de crear una nueva aplicación/modelo, incluso importar y exportarlas en formato *.json*. Las utilizaciones de cada modelo se pueden visualizar en la columna *Visitas de puntos de conexión*.

Ahora, se va a explicar el funcionamiento de cada parte que compone la aplicación **Luis**. Cada aplicación/modelo contiene la sección panel, crear, administrar, entrenar, prueba y publicar.

3.6.2. Panel

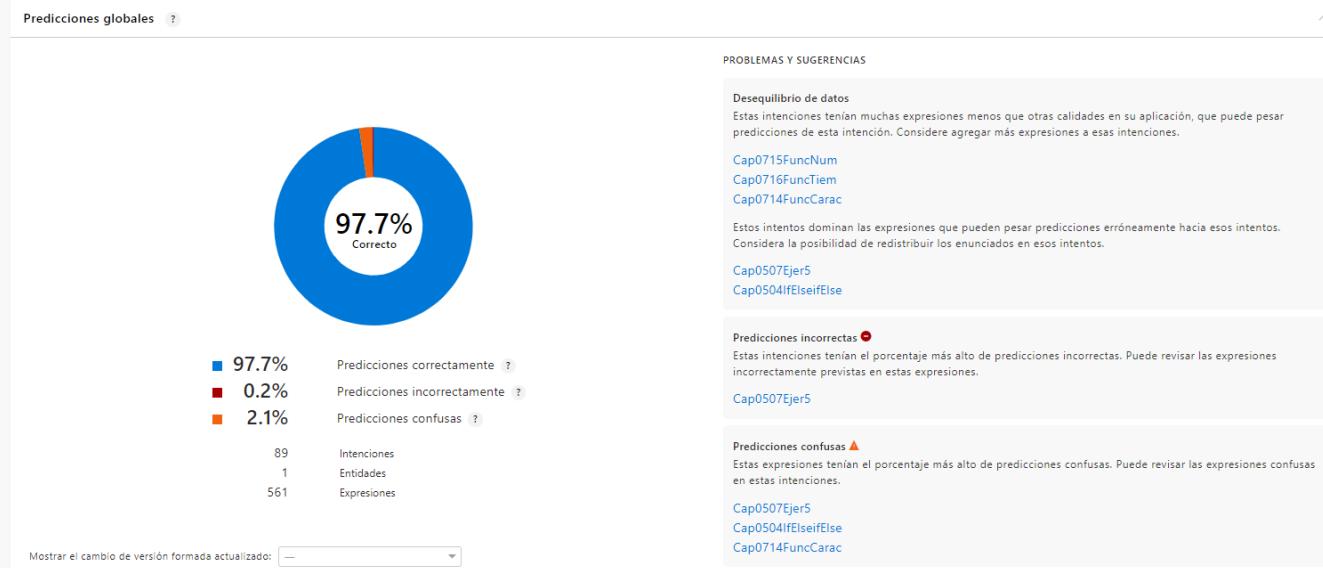
Aplicación publicada ?



Luis App 2.- Panel 1

Evaluación de la formación ?

Versión activa:0.1 – formadoMay 30, 2019, 8:10:37 PM

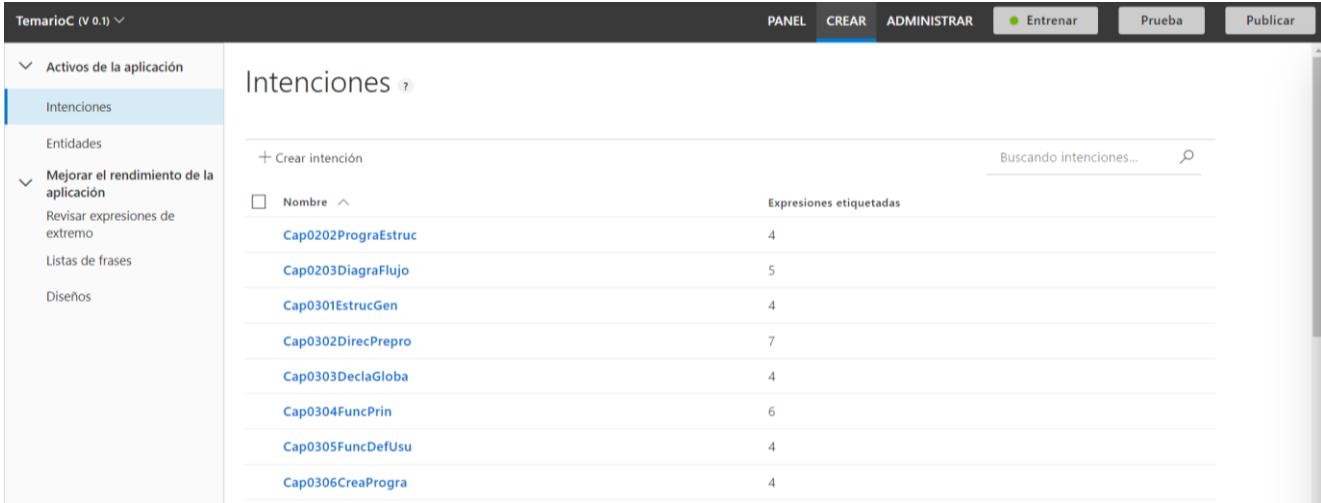


Luis App 3.- Panel 2

Según se muestra en Luis App 2.- Panel 1 y Luis App 3.- Panel 2:

El panel muestra la información de uso más relevante para cada modelo, los usos en el tiempo (por día) y la cantidad de predicciones correctas, incorrectas y confusas. Adicionalmente, muestra la cantidad de intenciones, entidades y expresiones que contiene. Además, proporciona información sobre algunos problemas y sugerencias para el modelo.

3.6.3. Crear



The screenshot shows the LUIS App interface with the 'Intenciones' tab selected. On the left, a sidebar menu under 'Activos de la aplicación' includes 'Intenciones', 'Entidades', and 'Mejorar el rendimiento de la aplicación'. The main area displays a table of intentions with columns for 'Nombre', 'Expresiones etiquetadas', and a search bar.

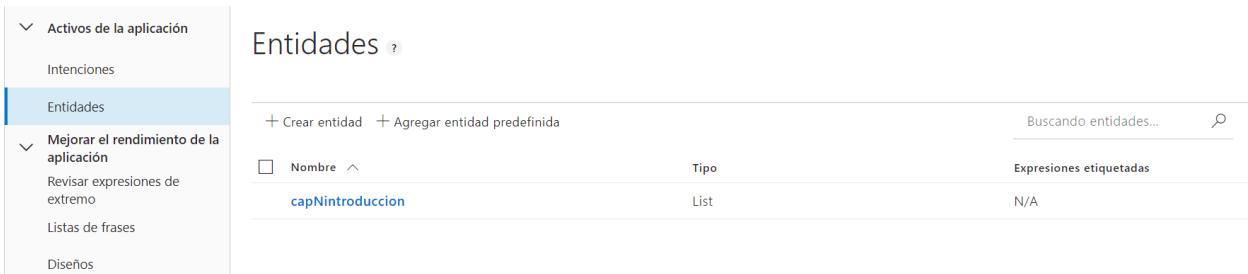
Nombre	Expresiones etiquetadas
Cap0202PrograEstruc	4
Cap0203DiagraFlujo	5
Cap0301EstrucGen	4
Cap0302DirecPrepro	7
Cap0303DeclaGloba	4
Cap0304FuncPrin	6
Cap0305FuncDefUsu	4
Cap0306CreaProgra	4

LUIS App 4.- Construir 1 – Intenciones

Según muestra LUIS App 4.- Construir 1 – Intenciones.

La pestaña *Intenciones (intents)* muestra todas las intenciones que recoge el modelo y las expresiones que forman parte de cada intención.

Por ejemplo, la intención “Cap0202PrograEstruc” contiene cuatro expresiones. *LUIS* comparará el mensaje de entrada con las expresiones que contenga y si la probabilidad más alta es con una de las cuatro anteriores se desencadenará la intención “Cap0202PrograEstruc”.



The screenshot shows the LUIS App interface with the 'Entidades' tab selected. On the left, a sidebar menu under 'Activos de la aplicación' includes 'Entidades', 'Intenciones', and 'Mejorar el rendimiento de la aplicación'. The main area displays a table of entities with columns for 'Nombre', 'Tipo', and 'Expresiones etiquetadas'.

Nombre	Tipo	Expresiones etiquetadas
capNintroduccion	List	N/A

LUIS App 5.- Construir 2 – Entidades

Según muestra LUIS App 5.- Construir 2 – Entidades.

La pestaña *Entidades, entities*, contiene las entidades utilizadas. Existen diferentes tipos de entidades (simple, compuesta, lista, jerárquica, expresión regular, patrón) y cada una sigue unas pautas diferentes en creación y funcionamiento.

Por ejemplo, la entidad “capNintroduccion” es de tipo lista y almacena los números de los capítulos que contienen el apartado introducción.

- ✓ Activos de la aplicación
- Intenciones
- Entidades
- ✓ Mejorar el rendimiento de la aplicación
- Revisar expresiones de extremo**
- Listas de frases
- Diseños

Revisar expresiones de extremo ?

Filtrar: Cap0202PrograEstruc Vista de entidades

Expresión	Intención alineada	Add/Delete
capítulo 5 introducción	Cap_Introduccio... ▾	
introducción capítulo capNintroduccion	Cap_Introduccio... ▾	
ejercicio 10	Cap0506Ejer4 (0... ▾	
situación variables locales y globales con especificadores static y auto capítulo 5	Cap0718SituVa... ▾	
capítulo capNintroduccion introducción	Cap_Introduccio... ▾	
capítulo cinco introducción	Cap_Introduccio... ▾	

LUIS App 6.- Construir 3 – Revisar expresiones de extremo

Según muestra LUIS App 6.- Construir 3 – Revisar expresiones de extremo.

LUIS posee una funcionalidad de retroalimentación o enriquecimiento que almacena los mensajes de entrada (expresiones) no reconocidos. Esto permite manualmente analizarlos y si es preciso incluirlos como nuevo conocimiento.

El mensaje de entrada (expresión) no reconocido se asocia a la intención con mayor coincidencia (aunque permite asociarlo a otra) para, si es preciso, añadirlo al instante.

La *Lista de frases* y los *Diseños* no se utilizan.

3.6.4. Administrar

Recoge toda la información útil para conectar la aplicación/modelo al Bot, como el identificador, el nombre, la cultura, la clave, el estado, la versión, etc y permite modificar algunos de esos valores.

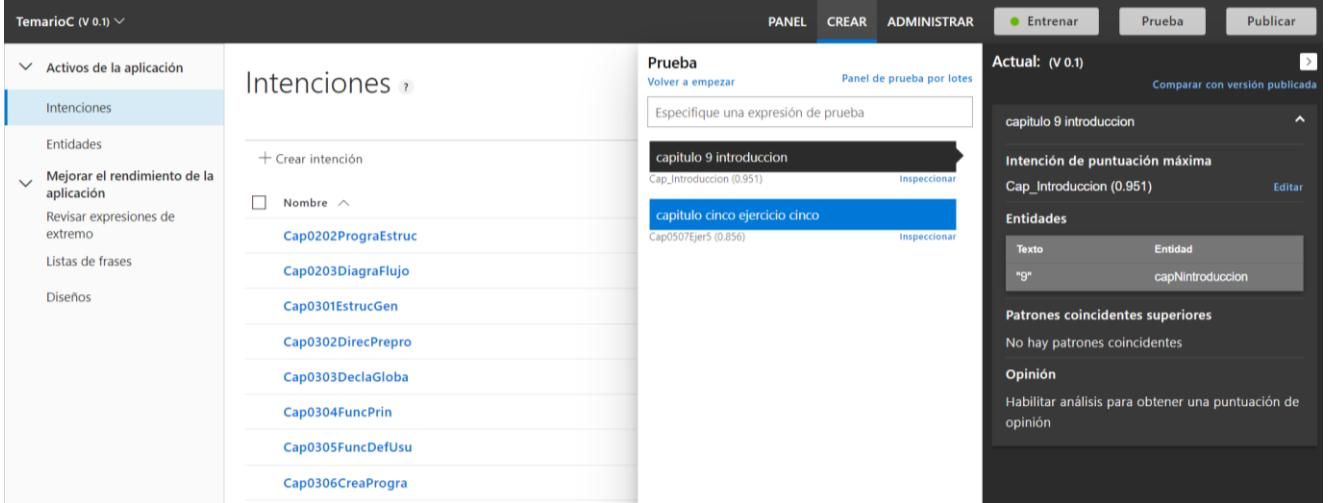
3.6.5. Entrenar

Según muestra LUIS App 4.- Construir 1 – Intenciones.

Cuando se crea una intención (con sus expresiones correspondientes) el modelo **LUIS** debe ser entrenado. Al pulsar **Entrenar**, la aplicación **LUIS** procesa la información y consigue la adaptación del modelo a dicha intención.

El indicador al lado del botón **Entrenar** dice si el modelo esta entrenado (verde) o no (rojo).

3.6.6. Prueba



The screenshot shows the LUIS App interface. On the left, a sidebar lists categories like 'Activos de la aplicación', 'Intenciones', 'Entidades', and 'Mejorar el rendimiento de la aplicación'. The main area is titled 'Intenciones' and lists several intents: 'capítulo 9 introducción', 'capítulo cinco ejercicio cinco', 'Cap0202PrograEstruc', etc. To the right, a panel titled 'Prueba' shows the input 'capítulo 9 introducción' and its associated entities: 'Cap_Introducción (0.951)' and 'capnIntroducción'. Below this, an 'Inspección' section shows the intent 'capítulo 9 introducción' highlighted in blue. A status bar at the top right indicates 'Actual: (V 0.1)'.

LUIS App 7.- Prueba

Según muestra LUIS App 7.- Prueba.

LUIS App permite hacer pruebas pulsando **Prueba** e introduciendo el mensaje de entrada correspondiente. Se muestra que intención se desata junto a su probabilidad, y las entidades que contiene.

Además, estas pruebas permiten corregir la intención desatada si existe ambigüedad entre el mensaje de entrada y las expresiones. ¡Ojo! antes de hacerlo, hay que buscar la manera de que no haya expresiones parecidas para intenciones distintas.

3.6.7. Publicar

Según muestra LUIS App 4.- Construir 1 – Intenciones.

Para que los cambios surtan efecto fuera de **LUIS** App (por ejemplo, en el Bot) hay que publicar cada modelo pulsando **Publicar**. Más concretamente, cuando se modifique, añada o elimine una *intención*, primero hay que **Entrenar** y posteriormente **Publicar** el modelo. Para *entidades*, cuando se cree o elimina una, solo se debe entrenar y publicar.

3.7. Conversión del Resumen de C a tarjetas adaptativas

Importante: Todas las herramientas utilizadas para la conversión del Resumen de C a tarjetas adaptativas están referenciadas en Herramientas de construcción.

El contenido sobre el APRENDIZAJE DE C se realizó de tal forma que sirviese para su posterior conversión a código procesable por el Bot, es decir, se diseñó con el objetivo de que tras su conversión mantuviese el mismo aspecto (títulos, párrafos, imágenes, notas y avisos, código, etc.).

3.7.1. Conversión de texto a código procesable

Para que el Bot pueda interpretar la información sobre el APRENDIZAJE DE C, todo el contenido debe ser convertido a código procesable y almacenada en *archivos de notación (.json)*. Por



consiguiente, cada punto ha seguido una serie de transformaciones que convierten un archivo *.docx* en un archivo *HTML*.

- 1^a conversión - *.docx* → MARKDOWN: Un documento de Word posee no solo texto plano sino también marcaje (negrita, cursiva, títulos, viñetas, ...) así que para mantenerlo hay que convertirlo primero a MARKDOWN.
- 2^a conversión - MARKDOWN → HTML: Ahora toca tratar los acentos, caracteres de extra, símbolos, etc o, lo que es lo mismo, convertir texto del paso anterior a texto *HTML*.

Mediante aplicaciones online se pueden conseguir ambos procesos de manera asequible. Asequible porque no todos los caracteres son convertidos y toca modificarlos a mano con ayuda de la tabla estándar UNICODE. Con esto se consigue convertir un archivo de texto con marcaje a texto *HTML* procesable por el Bot.

Ejemplo

Se va a utilizar el apartado del TEMARIO “Declaraciones globales” para ejemplificar el *desarrollo* de una conversión de texto a texto procesable.

Declaraciones globales

Las *declaraciones globales* indican al compilador que tanto las *funciones* como *variables globales* son comunes para todas las funciones del programa. Las declaraciones globales se sitúan antes de la función principal (*main*). Aclarar que cualquier función creada se considera función global para todo el programa.

```
#include <stdio.h>

int var_global;      //variable global para todo el programa

int funcion()        //funcion comun para todo el programa
{
|
| ...
}

int main()
{
|
| ...
}
```

Nota: Las variables locales se definen en cada función y solo son utilizables por esa función.

Ilustración 18.- Texto en formato .docx

Para comenzar, se coge la porción de texto a convertir, sin incluir imágenes, y se guarda en un nuevo *.docx* (Ilustración 18.- Texto en formato *.docx*). Entonces se ejecuta el convertidor

online referenciado en Conversión de texto Word a Markdown, que convierte .docx a MARKDOWN obteniendo un texto de salida con la simbología típica de MARKDOWN (Ilustración 19.- Texto en formato MARKDOWN).

Declaraciones globales

Las _declaraciones globales_ indican al compilador que tanto las _funciones_ como _variables globales_ son comunes para todas las funciones del programa. Las declaraciones globales se sitúan antes de la función principal (main). Aclarar que cualquier función creada se considera función global para todo el programa.

Nota: Las variables locales se definen en cada función y solo son utilizables por esa función.

Ilustración 19.- Texto en formato MARKDOWN

Ahora se coge el texto en formato MARKDOWN y se pasa por otro convertidor online referenciado en Conversión de texto a HTML que transforma cualquier texto a *HTML* (Ilustración 20.- Texto procesable), consiguiendo texto procesable (*HTML + MARKDOWN*) que podrá ser usado por el Bot sin problemas.

Declaraciones globales

Las _declaraciones globales_ indican al compilador que tanto las _funciones_ como _variables globales_ son comunes para todas las funciones del programa. Las declaraciones globales se sitúan antes de la función principal (main). Aclarar que cualquier función creada se considera función global para todo el programa.

Nota: Las variables locales se definen en cada función y solo son utilizables por esa función.

Ilustración 20.- Texto procesable

Nota: Algunos caracteres habrá que añadirlos manualmente desde el estándar Unicode, referenciado en Código UNICODE, en formato HTML por ejemplo.

3.7.2. Encapsulación del texto procesable en tarjetas adaptativas

La forma de exposición de la información más parecida a un libro dentro del marco Bot, son las *tarjetas adaptativas*. Así, cada punto del APRENDIZAJE DE C, ya en código procesable, ha sido introducido en dichas tarjetas.

Mediante la unificación de todo el APRENDIZAJE DE C en el sistema de *tarjetas adaptativas*, se ha conseguido en cada punto: una distinción clara y estructurada del contenido (bloques,

columnas, titulación, ...), exposición de imágenes y sugerencias, posibilidad de incluir preguntas y opciones, etc.

Las *tarjetas adaptativas* almacenan la información en *archivos de notación (.json)* y por tanto tienen un funcionamiento parecido a los demás archivos *.json* (pares clave-valor).

Ejemplo

Teniendo el texto procesable disponible (Ilustración 20.- Texto procesable), se va a proceder a crear la tarjeta adaptativa correspondiente, utilizando el diseñador de *tarjetas adaptativas* de Microsoft, referenciado en Herramientas de gestión de tarjetas adaptativas.

El diseño va a ser explicado paso a paso, dando una explicación clara de todo lo necesario, aunque solamente se van a explicar las funcionalidades que irán de la mano de dicho diseño, es decir, no se explicarán todas. Aun así, el conocimiento adquirido se podrá extraer al resto de funcionalidades.

Según se muestran en las imágenes Ilustración 21.- Inserción de un bloque de texto, Ilustración 22.- Inserción de una imagen e Ilustración 23.- Inserción de una columna con dos bloques de texto:

Al pulsar el botón “New card” se iniciará una tarjeta en blanco. Ahora, siguiendo la estructura del texto representado en la imagen Ilustración 18.- Texto en formato .docx, a grosso modo, habrá que añadir un bloque de texto, una imagen y una columna con dos bloques de texto:

- Bloque de texto.** Se insertará un bloque de texto arrastrando el botón “TextBlock” al diseño. Dentro de las propiedades del elemento, en el campo “Text”, se insertará el título y el primer párrafo del texto procesable (Ilustración 20.- Texto procesable). Además, se marcará el campo “Wrap” para que el texto se ajuste a la *tarjeta adaptativa*. Los demás campos no se tocarán, pero sirven para darle un estilo personalizado (color, tamaño, anchura, alineación, separación, etc).

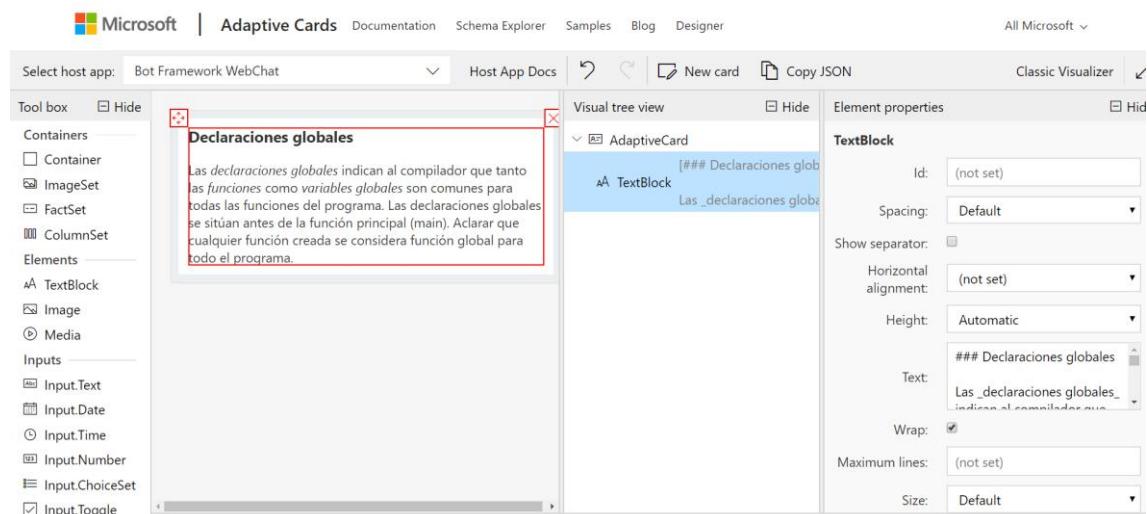


Ilustración 21.- Inserción de un bloque de texto

Antes de inserta la imagen, hay que generar una URL a la misma... Se puede pensar que cualquier almacén en la nube es capaz de proporcionar una URL, pero aquí surge el problema y es que no todos los almacenes procesan las imágenes, sino que solamente las almacenan. Para suplir esta carencia se ha utilizado imgbb que almacena y procesa las imágenes rápidamente además de contar con una interfaz sutil, atractiva y gratuita. Se hace referencia a él en Herramientas de depuración y prueba de Bots

- Emulador → **Bot Framework Emulator V4**

Muestra el comportamiento visual del Bot permitiendo hacer correcciones a nivel de interfaz y de texto. Además, junto a la herramienta de depuración de **VISUAL STUDIO 2017 COMMUNITY** permite realizar pruebas de código de manera exhaustiva. Es una herramienta imprescindible para la visualización y prueba de Bots en desarrollo.

Almacén de imágenes de las Herramientas de construcción del Bot.

2. **Imagen.** Primero, se introducirá una imagen arrastrando el botón “Image” al diseño. Ahora se insertará la URL de la imagen procesada correspondiente en el campo “Url”. Adicionalmente, insertando la URL en el campo “Url” de “Select action” marcando “Action.OpenUrl”, se le permitirá al Usuario abrir la imagen en la Web dándole un vistazo más detallado.

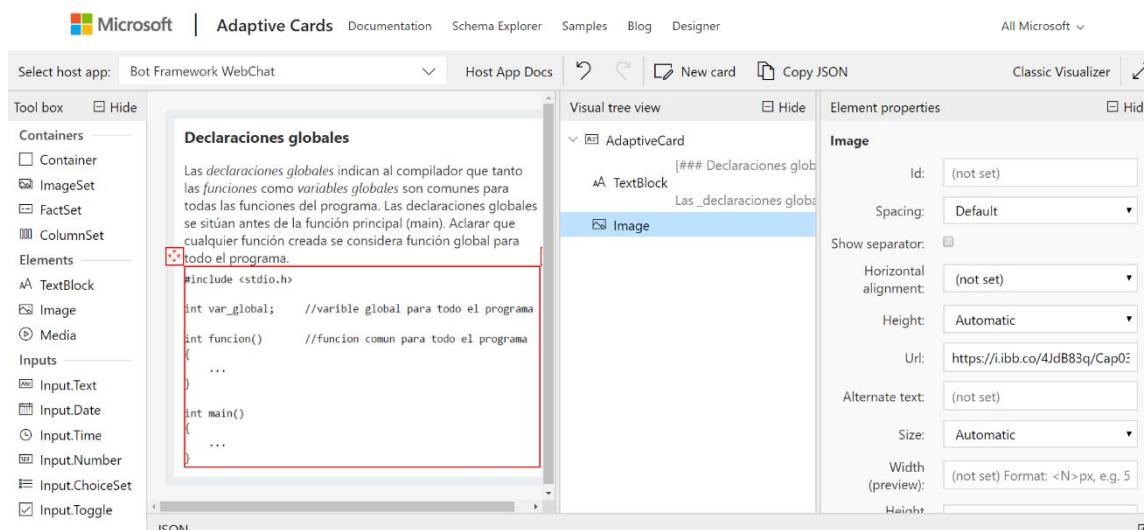
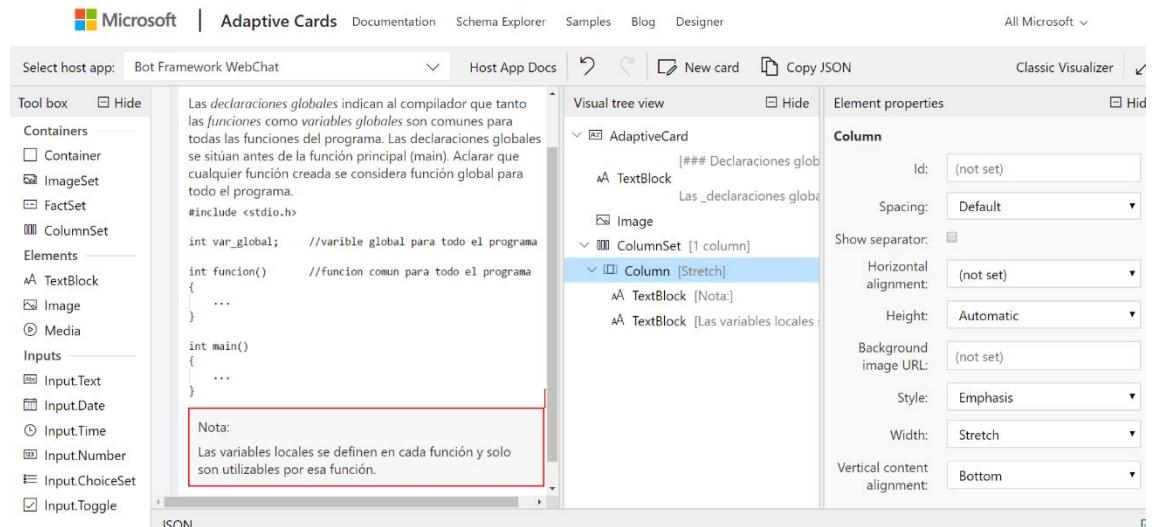


Ilustración 22.- Inserción de una imagen

En caso de sugerencias como notas, avisos, etc se hará un diseño de dos bloques dentro un bloque columna, permitiendo hacer énfasis del texto dentro de dicha columna.

3. **Columna con dos bloques de texto.** En primer lugar, se arrastrará el botón “ColumnSet” al diseño, en segundo lugar, se pulsará el botón “Add a column” que está justamente en la esquina inferior derecha del ColumnSet para añadir una sola columna y, por último, se arrastrarán dos bloques de texto dentro de la columna: uno para la palabra “Nota” y otro para el texto de la nota. Insertando un bloque columna es posible hacer énfasis, eligiendo la opción “Emphasis” del campo “Style” en la columna,

quedando resaltado.



The screenshot shows the Microsoft Adaptive Cards Designer interface. On the left, there's a 'Tool box' with various card components like Container, ImageSet, FactSet, ColumnSet, TextBlock, Image, Media, Input.Text, Input.Date, Input.Time, Input.Number, Input.ChoiceSet, and Input.Toggle. In the center, there's a 'Select host app:' dropdown set to 'Bot Framework WebChat', a 'Host App Docs' link, and several buttons: 'New card', 'Copy JSON', 'Classic Visualizer', and 'Hide'. To the right, there's a 'Visual tree view' pane showing the hierarchical structure of the AdaptiveCard, and an 'Element properties' pane for the selected 'Column' element. The JSON code in the center is:

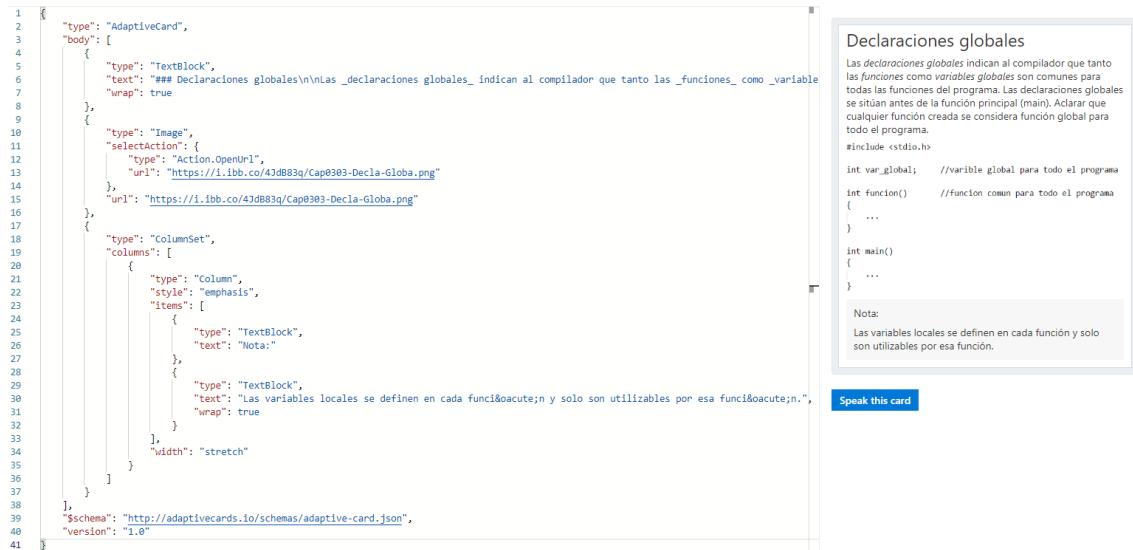
```

1 {
2   "type": "AdaptiveCard",
3   "body": [
4     {
5       "type": "TextBlock",
6       "text": "### Declaraciones globales\n\nLas _declaraciones globales_ indican al compilador que tanto las _funciones_ como _variables globales_ son comunes para todas las funciones del programa. Las declaraciones globales se sitúan antes de la función principal (main). Aclarar que cualquier función creada se considera función global para todo el programa."
7       "wrap": true
8     },
9     {
10      "type": "Image",
11      "selectAction": {
12        "type": "Action.OpenUrl",
13        "url": "https://i.ibb.co/4Jd83q/Cap0303-Decla-Globa.png"
14      },
15      "url": "https://i.ibb.co/4Jd83q/Cap0303-Decla-Globa.png"
16    },
17    {
18      "type": "ColumnSet",
19      "columns": [
20        {
21          "type": "Column",
22          "style": "emphasis",
23          "items": [
24            {
25              "type": "TextBlock",
26              "text": "Nota:"
27            },
28            {
29              "type": "TextBlock",
30              "text": "Las variables locales se definen en cada función y solo son utilizables por esa función."
31              "wrap": true
32            }
33          ],
34          "width": "stretch"
35        }
36      ]
37    }
38  ],
39  "$schema": "http://adaptivecards.io/schemas/adaptive-card.json",
40  "version": "1.0"
41 }

```

Ilustración 23.- Inserción de una columna con dos bloques de texto

Una vez estructurada la información (como en Ilustración 18.- Texto en formato .docx) y habiendo conseguido el estilo deseado, se pulsará el botón “Copy JSON” que copiará el código .json de la tarjeta creada. El código .json queda como se muestra en la imagen Ilustración 24.- Tarjeta Adaptativa (.json). Finalmente, se llevará dicho código al archivo de notación (.json) correspondiente.



The screenshot shows the Microsoft Adaptive Cards Designer interface with the JSON code from the previous screenshot. A modal window titled 'Declaraciones globales' is open, showing the rendered AdaptiveCard with its visual elements and text content. The modal includes a 'Speak this card' button at the bottom.

Ilustración 24.- Tarjeta Adaptativa (.json)

Nota: Para crear un estilo personal existe la opción de tocar los demás campos para bloques de texto, imágenes, columnas, etc. Por otro lado, también se puede modificar el estilo conociendo en profundidad el marcaje MARKDOWN. A parte se pueden añadir opciones de pregunta, videos, entrada de texto, de números, y muchas otras cosas.

4. Conclusiones y trabajos futuros

4.1. Conclusiones

BOTAPRENDIZAJEC es un Bot de tipo conversacional informativo, es decir, la interacción Usuario-Bot se centra en la enseñanza del lenguaje C básico y medio. Posee un comportamiento muy sencillo que mediante la introducción de texto corto o la pulsación de un botón desata la información contenida. El contenido abarca desde la explicación de la teoría hasta problemas y ejercicios de diversa índole que completan una enseñanza exhaustiva. Posee funcionalidades muy interesantes como cancelar la conversación dando la posibilidad de cambiar el rumbo del aprendizaje, mostrar la ayuda, continuar con el último punto consultado, ... pero su principal baza es la eficacia con la que muestra cada apartado del APRENDIZAJE DE C mediante títulos, texto, notas, avisos, viñetas, énfasis, imágenes, acceso a código C, etc etc de un modo claro, conciso y sin olvidar ningún detalle.

Las conclusiones se han dividido en una primera parte que muestra el funcionamiento general, en otra que muestra las sensaciones tenidas al utilizar la tecnología Bot, en concreto, al utilizar AZURE BOT SERVICE y, por último, en las mejoras que se podrían realizar tanto a nivel de la plataforma de Azure como del **BOTAPRENDIZAJEC** en particular.

4.1.1. Funcionamiento

Este apartado muestra el funcionamiento resumido de las características que posee el **BOTAPRENDIZAJEC**. Contiene la parte principal que describe el comportamiento de las funciones generales y da acceso al TEMARIO y los PROBLEMAS de C; la parte del TEMARIO que contiene su índice, las funcionalidades correspondientes y todo la teoría sobre el lenguaje básico y medio de C; la parte de los PROBLEMAS que contiene problemas de diversa índole, su índice y las funcionalidades correspondientes; el comportamiento pasivo, es decir, las funcionalidades que actúan de forma pasiva tras la entrada y salida de mensajería; los tipos de sugerencias que el Bot muestra, de la mano de las buenas prácticas de C, junto a un punto del APRENDIZAJE DE C; y las características que no se pudieron implementar.

Parte PRINCIPAL (Main)

- **Bibliografía.** Muestra la *Tarjeta de la Bibliografía* que incluye toda la información seguida para construir el Bot.
- **Crear nuevo usuario.** Se encarga de recoger los datos de usuario para más tarde almacenar su actividad (partes consultadas, último punto consultado, seguimiento de la conversación, etc). No implementado completamente.
- **Tarjeta de Introducción.** Muestra la *Tarjeta de Introducción*. La tarjeta recoge la IMAGEN (ícono) y el NOMBRE (identificador) del Bot, un texto de bienvenida y un texto corto del desempeño. Además, proporciona botones para un acceso rápido a las funciones más útiles como *índice del TEMARIO*, *índice de los PROBLEMAS*, *ayuda*, *bibliografía* y *agradecimientos*, *referencias a C*, ...
- **Tarjeta de Ayuda.** Muestra la *Tarjeta de Ayuda* y permite elegir que consultar. Las consultas abarcan Info sobre la parte del TEMARIO de C y PROBLEMAS de C, preguntas sobre como buscar en ambas partes, y otra información relevante.

- **Continuar por la última consulta.** Permite continuar por el último apartado/problema (si no se cierra el emulador).
- **TEMARIO de C.** Recoge cada apartado, subapartados y ejercicios de la parte del TEMARIO.
- **PROBLEMAS de C.** Recoge cada problema de la parte de los PROBLEMAS.
- **Respuesta confusa.** El Bot usa una respuesta de confusión (de un conjunto de respuestas de confusión) cuando no entiende algo. Si se ha alcanzado un máximo de respuestas confusas entonces se muestra una respuesta de sugerencia.
- **Desencadenadores de tarjeta.** Ejecuta el contenido que corresponde al botón pulsado desde una tarjeta (Tarjeta de Introducción, Tarjeta de Ayuda, ...).

Parte TEMARIO (teoría + ejercicios)

- **Índice TEMARIO.** Muestra el índice del TEMARIO y da una breve explicación sobre como buscar un punto.
- **Explicación de apartado.** Explica cada apartado o subapartado de un tema con texto, imágenes, énfasis, viñetas, etc. Se da opción de continuar o no con el siguiente.
- **Enunciado y solución ejercicio.** Muestra el enunciado del ejercicio correspondiente permitiendo al Usuario realizarlo por su cuenta. Se da la opción de mostrar la solución o pasar al siguiente apartado.
- **Contenido al siguiente capítulo.** Al finalizar un capítulo se da la posibilidad de mostrar el contenido del siguiente. Muestra de un vistazo los puntos que contiene el siguiente capítulo.
- **Cancelar y Ayuda.** Da la opción en cada apartado de abortar la continuación al siguiente, permitiendo consultar uno diferente. Si se quisiera consultar la ayuda, primero habrá que cancelar el diálogo activo.
- **Fin del TEMARIO.** Al finalizar el TEMARIO se da la enhorabuena y se recomienda continuar con los PROBLEMAS para reforzar la enseñanza y completarla.

Parte PROBLEMAS

- **Índice PROBLEMAS.** Muestra el índice de los PROBLEMAS y da una breve explicación sobre como buscar uno.
- **Problemas para realizar.** Muestra el enunciado del problema y todos los requisitos a seguir para su realización. Se da la opción de pasar o no al siguiente problema.
- **Problemas explicados.** Muestra el enunciado del problema dando la opción de realizarlo. A parte se la opción de mostrar la explicación o pasar al siguiente problema.
- **Problemas de solución desbloqueable.** Muestra el enunciado del problema además de una o varias cuestiones a responder. En función de los aciertos se mostrará un tipo de respuesta. Una vez se ha contestado hay opción de pasar al siguiente problema.
- **Cancelar y Ayuda.** Da la opción en cada problema de abortar la continuación al siguiente, permitiendo consultar uno diferente. Si se quisiera consultar la ayuda, primero habrá que cancelar el diálogo activo.
- **Fin de los PROBLEMAS.** Al finalizar los PROBLEMAS se da la enhorabuena y se recomienda continuar con ejercicios más extensos e incluso adaptados a la asignatura que imparte el profesor.

Comportamiento pasivo

- **Indicador de procesamiento.** El Bot hace una señal (indicador) al Usuario cuando está respondiendo/trabajando, por ejemplo, cuando está buscando la información solicitada.
- **Autoguardado de estados.** Tanto el estado de la Conversación como es estado de Usuario se guarda automáticamente en cada turno de conversación. Aunque se pierden al cerrar la aplicación Bot.
- **Fijación del idioma.** El idioma se fija en cada turno de conversación mediante la conexión a los servicios de Azure, aunque en este caso sigue la fijación por defecto “es-ES”.
- **Diálogos base.** El Bot sigue una estructura de diálogos base en los que se apoyan o de los que derivan todos los demás; a su vez los diálogos base derivan del diálogo de componentes (*ComponentDialog*). Estos diálogos base consiguen encaminar/encauzar las preguntas del usuario o actuar sobre las posibles interrupciones. Hay dos ramas bien definidas:
 - El diálogo encaminador (*RouterDialog*) que deriva en el diálogo *Main*.
 - Los diálogos de interrupciones (*InterruptableDialog* y *AprendizajeCDialog* respectivamente derivados) que derivan en cada punto/diálogo del TEMARIO y de los PROBLEMAS.

Tipos de sugerencias:

Dentro del Bot en general y principalmente dentro de la información mostrada sobre el APRENDIZAJE DE C, se proporcionan sugerencias de uso para encaminar al Usuario en las buenas prácticas de C.

- **Notas.** Información sugerida que se nutre de las buenas prácticas de C (color de fondo azul claro).
- **Avisos.** Advierten de errores comunes de C (color de fondo azul oscuro).
- **Restricciones.** Limita el uso de alguna funcionalidad de C en determinados problemas (color de fondo naranja claro).

Características no implementadas:

- *Redirección rápida a los ejercicios de cada capítulo en cada apartado y subapartado de capítulo.* No implementada por crear una respuesta lenta en el Bot (debido a la inclusión de un dialogo más en el diálogo de cascada).

4.1.2. Sensaciones

A continuación, se van a redactar las sensaciones que se han tenido al programar un Bot en general y mediante AZURE BOT SERVICE en particular, a parte de algunas otras cuestiones.

Uso de la tecnología Bot

Los Bots en general son sistemas muy útiles para llevar a cabo múltiples tareas, pero sin duda los Bots más interesantes son los de tipo conversacional. Prácticamente todas las ventajas y oportunidades que ofrece una aplicación Web pueden ser incluidas en Bots (de hecho, los Bots derivan de aplicaciones Web) y sumado a la posibilidad conversacional y a los servicios externos como servicios cognitivos, sin duda son sistemas a tener en cuenta.

Pueden abarcar terrenos como el de la gestión, la enseñanza, las compras, la interoperabilidad entre empresas y sectores, asistencia de todo tipo, y mucho más. Más aun, junto al internet de las cosas, IoT, pueden llegar a cualquier lugar y dispositivo electrónico pudiendo convertirse en su infraestructura principal. Su alcance y validez son cada vez más positivos en el mundo actual.

Uso de AZURE BOT SERVICE

La plataforma de Microsoft que da soporte, gestiona, permite crear, desarrollar, depurar y emular Bots es **AZURE BOT SERVICE**. Ha sido empleada para construir el **BOTAPRENDIZAJE** utilizando solamente los servicios gratuitos que ofrece, los cuales son suficientes para construir un Bot complejo.

Es una plataforma potente con cierto recorrido y madurez, que en este momento de su etapa está continuamente corrigiendo errores de funcionamiento, añadiendo nuevas funcionalidades y reinventando el **BOT FRAMEWORK** para ser más asequible. Debido a su madurez media no permite toda la funcionalidad que, de algún modo, se han echado en falta.

En contraposición, el versionado continuo que introduce nuevas funcionalidades, quita otras, haciendo que el programador tenga que estar con cierta continuidad adaptando el código a las nuevas características y por supuesto estudiando su funcionamiento.

Con el transcurso de las versiones, sin duda alguna, se ha ido consiguiendo un código mucho más claro, estructurado y centrado en las funcionalidades que un Bot precisa. Se puede decir que a lo largo de las versiones se está empezando a crear una verdadera plataforma para Bots desligando de alguna manera esa parte de aplicación Web, es decir, los Bots están empezando a conseguir personalidad propia como aplicación única, estando cada vez menos influenciados por el marco aplicación Web. Una analogía seria la evolución del lenguaje C, pasando a C++ y, a groso modo, a Java teniendo ambos personalidad propia, aunque su base sea C.

Versionado

Las versiones de **AZURE BOT SERVICE** que se han consultado en este proyecto han sido la v3.0, v4.2 y v4.4. Aunque la versión 3.0 se ha visto muy por encima se puede decir que hay un cambio sustancial con respecto a la 4.2 en prácticamente toda la arquitectura del Bot. Algunos de los cambios se sitúan en:

- **Flujo de entrada de mensajería.** Se pasa de un flujo de atributos junto a métodos a un flujo de solamente métodos clave muchos más compresible, es decir, ahora los protagonistas son los métodos clave no los atributos. Los métodos clave comienzan a definir una estructura de funcionamiento más compresible y con menos carga de código.
- **Comportamiento de los servicios externos.** Los servicios externos se anclaban al Bot de manera muy compleja, utilizando de nuevo demasiados atributos para desencadenarlos. Con la actualización se han creado clases estándar que recogen los servicios más utilizados, proporcionando una serie de métodos de uso mucho más comprensibles y rápidos.

- **Puesta en marcha.** Se ha pasado de un simple punto de entrada de mensajería que encauzaba la conversación hacia un punto determinado, si era preciso, cargando y utilizando el servicio correspondiente; a una puesta en marcha en toda regla -que inicializa los servicios utilizados por el Bot y los inserta en cualquier clase mediante inyección de dependencias- sumada al punto de entrada de mensajería.
- **Diálogos.** Se ha conseguido una estandarización en los diálogos pasando de diálogos a secas en los que el programador tenía que construirlos completamente a un estándar con tres tipos: un agrupador de diálogos (*diálogo de componentes*), un diálogo escalonado (*diálogos en cascada*) y un diálogo para pedir información al Usuario (*diálogo de petición*).
- **Cascada de diálogos.** Se ha pasado de métodos que llaman a otros métodos simulando una cascada a una verdadera clase *diálogo en cascada* que agrupa el comportamiento anterior en una sola clase.

Ahora tras el paso de la versión 4.2, en la cual se ha realizado el Bot, a la 4.4 se han conseguido estandarizaciones importantes, cambios en otros aspectos y algunos añadidos.

- Primero, en la versión 4.4 se ha pasado a controlar el Bot mediante MVC (Modelo Vista Controlador) manteniendo la independencia del almacén de información y la información mostrada al Usuario a través de un controlador. En la versión 4.2 se hacía de forma implícita y no se usaba un controlador en sí. Curiosamente en la versión 3.0 si se usaba un controlador como eje central, así que, no se sabe que rumbo tomará el Bot en este aspecto.
- Segundo, estandarizaciones importantes en los métodos clave:
 - Se ha estandarizado el control de interrupciones estando disponible para cada mensaje entrante.
 - Las acciones de botón se procesan como eventos y son consideradas eventos junto a los eventos del sistema.
 - Los mensajes escritos desembocan en un método propio que puede ser usado para realizar determinadas acciones, por ejemplo, puede encargarse de usar el servicio de reconocimiento del lenguaje.
 - El mensaje de bienvenida se procesa en un método propio.
 - Tras completar un diálogo, existe en un método propio que permitirá crear cualquier acción que se ajuste al funcionamiento buscado.
- Tercero, ahora todos los diálogos derivan del *diálogo de componentes* directamente, salvo el *Main* que deriva de una serie de clases que le permiten realizar todas las gestiones nombradas en el punto anterior. En la versión 4.2, el *Main* se derivaba de forma parecida, pero, el resto de los diálogos, se derivaban de clases que controlaban las interrupciones.
- Cuarto, se han añadido nuevos métodos que permite controlar: los eventos token (como los de autenticación de Usuario), los miembros que se desconectan del Bot y las actividades no reconocidas.

¿Y si no existiese AZURE BOT SERVICE?

En caso de no existir el servicio de Bots de Azure, habría que echar mano de las herramientas de creación de aplicaciones Web, prácticamente no existiría ninguna otra posibilidad. La base

para Bots de Azure parte del funcionamiento de aplicaciones Web, por lo tanto, se podrían crear Bots a partir de esta.

De hecho, las funcionalidades de aplicaciones Web están muy extendidas y perfectamente se podría crear un Bot complejo, la gran dificultad estaría en cómo crearlo partiendo de que las aplicaciones Web no poseen en sí la capacidad de conversación, además de que habría que adaptar dichas funcionalidades al comportamiento.

En resumen, solo la creación de un Bot sin el uso de **AZURE BOT SERVICE** mediante las herramientas de aplicaciones Web, sería una tarea compleja que requeriría, como mínimo, de otro proyecto para llevarla a cabo.

4.1.3. Mejoras

A continuación, se van a redactar las mejoras que sería aconsejable realizar tanto en el **BOT FRAMEWORK** como en el **BOTAPRENDIZAJE**, eso sí, siempre bajo el punto de vista de que el Bot sea conversacional de tipo informativo y dedicado a la enseñanza.

En el BOT FRAMEWORK

Algunos aspectos que se han echado de menos en cuanto al marco Bot han sido:

- Nuevos tipos de escritura: la posibilidad de elegir entre varios tipos de escritura, color y tamaño. Tener un solo tipo hace muy limitada la creación de una interfaz personal.
- Nuevas funcionalidades en *tarjetas adaptativas*: aumento de sus funcionalidades como cambios de color en el texto, tipos de escritura, poner una anchura mínima de tarjeta, posibilidad de crear tablas, poseer iconos básicos como el de ayuda y poseer colores de fondo básicos. Las *tarjetas adaptativas* están en un periodo muy verde de maduración (versión 1.2) así que con el paso de las versiones mejoran sustancialmente sus funcionalidades.
- Unificar las *tarjetas adaptativas* como un sistema estándar: hacer de las *tarjetas adaptativas* un sistema estándar que se plasme y funcione del mismo modo se cual sea el tipo de canal utilizado.
- Unificar los eventos del sistema: que actualmente están muy desestructurado, explicando la funcionalidad de cada uno de forma detallada. Incluir una funcionalidad controladora de tiempos que permitiese utilizar cuentas atrás y/o cronómetros de forma sencilla.

En el BOTAPRENDIZAJE

Teniendo en cuenta todos aspectos anteriores, a nivel del **BOTAPRENDIZAJE** sería interesante mejorar:

- Mejora de la interfaz de Usuario: mediante nuevos tipos de escritura y maduración de las *tarjetas adaptativas* se podría conseguir una interfaz más personal y adaptada a la temática del Bot.
- Adaptación a la versión 4.4 de **AZURE BOT SERVICE**: la nueva versión simplifica en gran medida el código necesario para lanzar un Bot complejo frente a la versión anterior.

Permite poner en marcha un Bot en menos tiempo y precisa de revisar menos código ante errores de funcionamiento.

- Modificaciones o restructuración en los archivos contenidos el TEMARIO y los PROBLEMAS de C: en su momento se creyó que la estructura en árbol seguida para empaquetar todo el contenido de C era la correcta. Aunque este tipo de estructura no está mal ni mucho menos... sí que sería bueno un cambio que hiciera que los archivos fuesen más accesibles abriendo el número mínimo de carpetas. Teniendo, ahora sí, una vista de todo el conjunto que supone este archivado, sería más sencillo encaminarlo hacia una estructura más centrada en carpetas para cada tipo de archivo. Es decir, convertir la estructura en árbol, que contiene las partes, capítulos, apartados y subapartados, a una estructura con una carpeta para cada tipo de archivo (como archivos de código (.cs), archivos de recursos (.resx) y *archivos de notación (.json)*).
- Inclusión del idioma inglés: el Bot tendría un tirón significativo si se le implementase el idioma inglés, pudiendo llegar a cualquier parte del mundo. Aunque el idioma español también posee su nicho de población, dar soporte al idioma inglés sería un gran avance que podría hacer crecer el Bot sustancialmente.

4.2. Trabajos futuros

Para terminar con la memoria del TFG es recomendable hablar sobre el futuro del Bot, es decir, que funcionalidades, cambios, reestructuraciones, nuevas implantaciones, adaptaciones, etc puede sufrir.

4.2.1. Versionado continuo del software

- **Adaptación a las actualizaciones.** Adaptación a las nuevas versiones de AZURE BOT SERVICE.
- **Arreglos y diseño visual.** Modificación de diseño visual (inclusión de Iconos en vez de palabras), del APRENDIZAJE DE C y de código interno.

4.2.2. Corto alcance

- **Lanzamiento en la nube.** Despliegue en la nube para poner el Bot en producción y poder lanzarlo al exterior.
- **Implementación de Canales.** Ajustar el contenido del Bot a los diferentes canales que se pretendan utilizar.
- **Servicios QnA.** Añadir servicios QnA para hacer al Bot más amigable (dicharachero).
- **Implementación de otros servicios cognitivos útiles.** Como:
 - *Moderador de contenido.* Para filtrar blasfemias y texto no deseado.
 - *Traductor de texto.* Visualizar y comprobar cómo trabaja el traductor para una posible implementación junto a otros idiomas.
 - *Corrector ortográfico.* Corrige el texto de entrada automáticamente para ayudar al Bot a procesar el mensaje de entrada del Usuario.
- **Implementación del lenguaje avanzado de C.** Ampliar la enseñanza de lenguaje C añadiendo estructuras, memoria dinámica, pilas, colas y árboles.

4.2.3. Medio alcance

- **Autenticación y almacenamiento.** Autenticación del Usuario para almacenar el progreso y sus datos (estado de la Conversación y del Usuario).



- **Ampliar el número de actividades sobre el lenguaje C.** Incluir nuevas actividades a las ya incluidas (ejercicios y problemas) a otra parte del Bot, por ejemplo, “Actividades” que tengan otro cometido distinto.

4.2.4. Largo alcance

- **Historial de navegación.** Resulta útil saber cuáles fueron los últimos apartados consultados, sobre todo si se ha alternado entre apartados del TEMARIO y los PROBLEMAS.
- **Temporizador.** Incluir un temporizador para un estudio ajustado a un tiempo.
- **Ingles.** Dar soporte al idioma inglés.
- **Elección de idioma.** Dar la posibilidad al Usuario de elegir el idioma que más le convenga (esta opción pondrá la herramienta del middleware, *SetLocaleMiddleware*, en parada).

4.2.5. Paralelismo

- **Enseñanza del lenguaje C++.** Aumento de la temática a la que está orientado el Bot principal permitiendo ahora enseñar el lenguaje C++.

Bibliografía

Libros y Sitios Web

Temario de C

Libro

PROGRAMACIÓN EN C. Luis Joyanes Aguilar, Ignacio Zahonero Martínez

tutorialspoint

Sitio Web: <https://www.tutorialspoint.com/cprogramming>

cppreference

Sitio Web: <https://es.cppreference.com/w/c>

Problemas de C

Libro

INFORMÁTICA APLICADA. PROGRAMACIÓN EN LENGUAJE C. Pedro María Alcover Garau, UPCT

Aprendizaje de C#

Microsoft

Página principal: <https://docs.microsoft.com/es-es/dotnet/#pivot=docs&panel=getstarted>

Guías de programación de C#: <https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/>

Tutoriales de C#: <https://docs.microsoft.com/es-es/dotnet/csharp/tutorials/index>

tutorialpoints

Sitio Web: <https://www.tutorialspoint.com/csharp/>

Aprendizaje sobre el Bot

Información general de Bot: <https://docs.microsoft.com/es-es/azure/bot-service/bot-service-overview-introduction?view=azure-bot-service-4.0>

Azure Bot Service

Sitio Web: <https://docs.microsoft.com/es-es/azure/bot-service/?view=azure-bot-service-4.0>

Anexo

Contenido del TEMARIO

El contenido del TEMARIO se ha introducido en el Bot tal cual se expresa a continuación -en *tarjetas adaptativas*- partido en puntos, apartados y subapartados. Se ha pasado de formato a .docx a formato .json como se muestra en Conversión del Resumen de C a *tarjetas adaptativas*. Agradecer de nuevo a los libros y sitios Web de donde se ha extraído, resumido, convertido e incluso modificado esta información descritos en la Bibliografía.

Temario. Objetivos, características y organización del contenido

Orientación y objetivos

Este Bot está pensado para poder servir de referencia para un primer curso de *introducción a la programación*, con una segunda parte que servirá como introducción a *estructuras de datos*, todo ello utilizando la versión estándar ANSI de C como lenguaje de programación. A la vez que se describe la sintaxis de C, se enseñarán técnicas de programación estructurada. Los objetivos son:

- Introducción a las ciencias de programación usando el estándar ANSI de C.
- Análisis, construcción y diseño de programas.
- Resolución de problemas mediante técnicas de programación.
- Enseñanza de las reglas de sintaxis más frecuentes y eficientes.

De esta manera se enseñarán técnicas clásicas y avanzadas de programación estructurada.

Características del contenido

La información proporcionada sigue en principio los siguientes elementos clave para conseguir el mayor rendimiento en el aprendizaje de C:

- **Índice: TEMARIO.** Muestra el índice del TEMARIO dividido en partes, capítulos y apartados.
- **Contenido de cada capítulo.** Enumera los apartados descritos en el capítulo.
- **Introducción.** Hace de introducción al capítulo describiendo los apartados iniciales del mismo.
- **Apartados: Explicación.** Explica cada apartado de un capítulo de forma breve pero eficaz.

- **Ejercicios: Enunciado y solución.** Muestra el enunciado del ejercicio correspondiente y da la opción de mostrar la solución.

Por otro lado, se puede incluir información extra como consejos al usuario, advertencias y reglas de uso, y técnicas de programación.

Organización del contenido

La información proporcionada se divide en dos partes que juntas constituyen la programación básica y media de C; dejando a un lado la programación avanzada no necesaria para este cometido. Las partes incluyen: **Metodología de programación, Fundamentos de programación en C.**

Se comienza con una introducción a la informática y a las ciencias de computación para confluir en la programación, consiguiendo un primer acercamiento al *lenguaje estructurado*. El aprendizaje es acumulativo, por lo que los primeros capítulos proporcionarán fundamentos conceptuales y ejercicios sencillos, seguidos de capítulos posteriores de modo progresivo que entrarán en más detalle.

PARTE I. METODOLOGÍA DE LA PROGRAMACIÓN

Comprende los conceptos básicos para el análisis, diseño y construcción de programas y sienta las bases del lenguaje C. Describe los elementos básicos y las técnicas de programación que se deberán emplear. Incluye los capítulos: **Capítulo 1. Introducción a la ciencia de la computación y a la programación** y **Capítulo 2. Fundamentos de programación**.

PARTE II. FUNDAMENTOS DE PROGRAMACIÓN EN C

Comprende la sintaxis, reglas y criterios de construcción del lenguaje C junto con temas específicos como punteros, arrays, cadenas, etc. Incluye los capítulos: **Capítulo 3. Elementos básicos del lenguaje C, Capítulo 4. Operadores y expresiones, Capítulo 5. Estructuras de selección: sentencias if y switch, Capítulo 6. Estructuras repetitivas: bucles (for, while y do-while), Capítulo 7. Funciones, Capítulo 8. Arrays (listas y tablas), Capítulo 9. Cadenas.**

PARTE I. METODOLOGÍA DE LA PROGRAMACIÓN

Capítulo 1. Introducción a la ciencia de la computación y a la programación

Introducción

Una computadora funciona a través de *programas*, los cuales le dan el código y las herramientas necesarias para funcionar. Según el código del programa, tendrá una función particular. Dicho código estará escrito en un *lenguaje de programación* que junto a las herramientas del lenguaje harán la función para la cual se ha creado. Por lo tanto, el *lenguaje de programación* nos sirve de base para la creación de cualquier programa. El *programador* será el encargado de escribir y diseñar dicho programa.

Existen tres tipos de lenguaje de programación ordenados por nivel de profundidad en la escritura: Lenguaje máquina, lenguaje de bajo nivel (ensamblador) y lenguaje de alto nivel. Nosotros nos centraremos en los lenguajes de alto nivel por ser más entendibles/intuitivos, portables a otro sistemas y sobre todo más útiles/eficientes a la hora de programar. El *compilador* será el traductor del programa fuente escrito en alto nivel, hacia el lenguaje máquina (hardware) creando un programa objeto. Algunos lenguajes de alto nivel son C, C++, C#, Visual BASIC, Java, ... pero nosotros nos centraremos en la enseñanza de **C**.

Capítulo 2. Fundamentos de programación

Introducción

Cuando sepamos que una determinada tarea se puede resolver mediante programación, estaremos listos para crear un programa que resuelva esa tarea. La tarea se podrá programar de una manera creativa o siguiendo una serie de pasos que generalmente debe seguir cualquier programador. Las fases o pasos son los siguientes: Análisis del problema, diseño del algoritmo, codificación, ejecución-verificación-depuración, mantenimiento y documentación.

- **Análisis del problema:** según los criterios requeridos y la solución deseada, se darán las pautas a seguir para la creación del programa. Es decir, mediante las entradas impuestas debemos diseñar el método que mejor se adapte para conseguir la solución.
- **Diseño del algoritmo:** ¿Cómo hacemos el programa? El método más eficiente es la *programación modular* que consiste en crear subprogramas, es decir, encaminar el programa dividiéndolo en partes más pequeñas hasta conseguir su resolución. Entonces habrá un programa principal que llamará a subprogramas que a su vez podrán llamar a otros subprogramas. Existen herramientas de programación que ayudan a resolver el programa de forma gráfica y son los diagramas de flujo y el pseudocódigo entre otros.
- **Codificación:** mediante el algoritmo de la fase de diseño y utilizando un lenguaje de programación específico se creará el programa.
- **Ejecución-verificación-depuración:** el programa en sí se denomina *programa fuente* o *código fuente*, que mediante el compilador y no habiendo ningún error pasa a ser *programa objeto*, no siendo ejecutable aún. Con el enlace (*link*) del programa objeto y las bibliotecas correspondientes para crear dicho programa se creará el *programa ejecutable*. Ahora se podrá ejecutar y ver el resultado. La verificación y depuración se encargará de encontrar errores, corregirlos y eliminarlos o darnos un aviso del error.
- **Mantenimiento y documentación:** es toda la información perteneciente al programa y que permite su comprensión. Entre ella se encuentra el análisis, los diagramas de flujo, pseudocódigo, manual de usuario, ... y los comentarios que incluya el código del programa.

Programación estructurada

Es una combinación de programación modular y *estructuras de control*. Hace que los programas sean más fáciles de escribir, verificar, leer y mantener. Una estructura de control es una estructura marcada con una función específica que permite modificar el flujo de ejecución de las instrucciones. Los tres tipos básicos son secuencia, selección y repetición.

Diagrama de flujo

Un diagrama de flujo es una manera muy útil de llevar acabo la creación de un programa de una forma gráfica, consiguiendo estructurarlo y entenderlo más fácilmente. En general, los diagramas de flujo son útiles durante la fase de aprendizaje del lenguaje, y ayudan tanto al diseño como a la revisión del algoritmo antes de programarlo.

Un *diagrama de flujo* es un diagrama que utiliza los símbolos estándar (cajas) unidos a flechas de dirección (líneas de flujo) para desarrollar algún tipo de proceso o programa.

Símbolo	Nombre	Función
	Inicio/Final	Representa el inicio y el final de un proceso
	Línea de Flujo	Indica el orden de ejecución de las operaciones
	Entrada/Salida	Lectura y escritura de datos
	Proceso	Operaciones en general
	Decisión	Toma de decisiones en base a verdadero/falso
	Subproceso	Subproceso de un proceso

Ilustración 25.- Diagrama de flujo

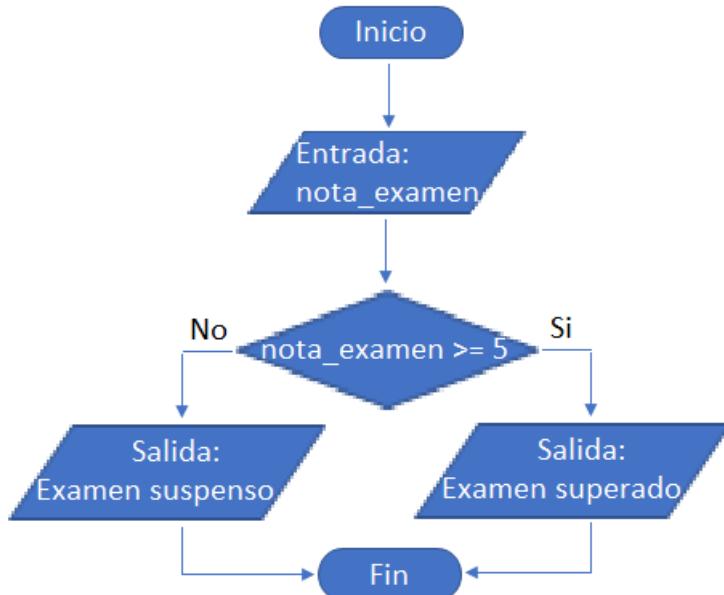


Diagrama de flujo 1.- Ejemplo 1

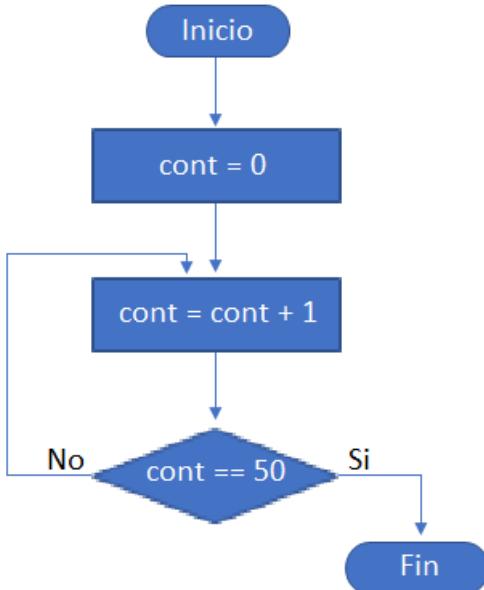


Diagrama de flujo 2.- Ejemplo 2

PARTE II. FUNDAMENTOS DE PROGRAMACIÓN EN C

Capítulo 3. Elementos básicos del lenguaje C

Estructura general

Por norma general un programa de C consta de una o más funciones, es decir, siempre contendrá la función *main* que se considera la función principal por donde comenzará y finalizará la ejecución del programa. Además, incluirá archivos de cabecera (*#include*), para añadir funciones definidas en otros ficheros, u otras directivas del preprocesador para definir funcionalidad extra.

```

#include <stdio.h> //directiva o archivo de cabecera

int main()          //funcion principal
{
|   ...            //contenido del main
}
    
```

Por otro lado, hay otras partes que también se consideran de gran importancia:

```
#define          //macros del preprocesador

Variables globales //usadas por la función main y demás funciones

tipo funcion()    //función
{
|
| ...
|}
```

Por lo tanto, un programa de C principalmente puede incluir: directivas del preprocesador (archivos de cabecera), declaraciones globales, la función *main*, funciones específicas, comentarios del programa, además de código utilizado para su realización.

```
/* Programa que imprime "hola mundo" */
#include <stdio.h>

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

Ejemplo 1.- Programa que imprime "Hola mundo"

Según el *Ejemplo*:

- **/* ... */** : Comentario. No será procesador por el compilador de C.
- **#include <stdio.h>** : Comprende la información relativa al estándar de funciones que gestionan la entrada y salida de datos de C, como es el caso de *printf()*. Los símbolos < y > indican que la directiva (librería) proviene de las librerías incluidas en el lenguaje C.
- **int main()** : la función *main* es la función donde se recoge todo el código, es decir, el código exterior a la función main debe ser llamado por esta para poder funcionar. Además, el contenido de la función main y demás funciones está encerrado entre llaves {...}.
- **printf()** : sentencia encargada, en este caso, de imprimir por pantalla el comentario entre comillas dobles.
- **return** : es una palabra reservada del lenguaje C que se encarga de devolver un valor (int, char, double, ...) y finalizar la función específica. El valor devuelto depende de la ejecución de la función.

Por último, hacer algunas aclaraciones. Los punto y coma (;) dentro de la función main indican al lenguaje C los finales de cada sentencia, es decir, hasta donde C tiene que leer una sentencia y a partir de los cuales tiene que comenzar a leer una nueva. El símbolo \n, dentro de un comentario entre comillas inserta un retorno de carro, que hace que el cursor de impresión por pantalla salte a una nueva línea.

Directivas del preprocesador

Las directivas del preprocesador incluyen `#include` y `#define`, además NO terminan en `();` como les ocurre a las sentencias. El símbolo almohadilla (`#`) indica al compilador que lea e incluya el contenido de esas directivas en dicha posición. Esta sintaxis, distinta a la del lenguaje C, se debe a que el preprocesador no es estrictamente parte del conjunto de herramientas del lenguaje.

Su uso más común son los *archivos de cabecera* como por ejemplo `stdio.h`, `stdlib.h`, `math.h` y `string.h` incluidos entre caracteres mayor-menor `<...>`, además de los *archivos cabecera creados por el usuario* que irán entre comillas “`...`” y serán buscados en la carpeta en la que se encuentra el programa creado.

```
#include <archivo_estandar_de_C.h>
#include "archivo_creado_por_el_usuario.h"
```

Nota: cuando se instala el compilador de C, automáticamente se instalan las librerías estándar de C.

`#include <stdio.h>`: incluye las funciones de entrada y salida de datos, ya sea por pantalla, por archivo y por flujos en el propio programa, como por ejemplo `scanf()`, `printf()`, `puts()`, `gets()`, `fprintf()` y `fscanf()`.

`#include <string.h>`: incluye funciones relacionadas con el tratamiento de cadenas de caracteres, como por ejemplo `strcmp()` que compara una cadena de caracteres con otra.

La directiva del preprocesador `#define` permite crear constantes u operaciones para el programa, como por ejemplo (`#define numero_PI 3.141516`).

```
#define numero_PI 3.1415926
```

Declaraciones globales

Las *declaraciones globales* indican al compilador que tanto las *funciones* como *variables globales* son comunes para todas las funciones del programa. Las declaraciones globales se sitúan antes de la función principal (`main`). Aclarar que cualquier función creada se considera función global para todo el programa.



```
#include <stdio.h>

int var_global;      //variable global para todo el programa

int funcion()        //funcion comun para todo el programa
{
|
| ...
}

int main()
{
|
| ...
}
```

Nota: Las variables locales se definen en cada función y solo son utilizables por esa función.

Función principal (main)

La función principal (*main*) habitualmente se situará a continuación de los archivos de cabecera y los prototipos de las funciones, y por encima de las funciones. Las llaves delimitan el bloque de sentencias que engloba esta función. Un programa solo puede tener una función *main*, en caso contrario habría un error de compilación.

```
int main()
{
|
| ...
}
```

Un programa corto puede incluir todas las sentencias en la función *main*, sin embargo, un programa extenso estaría dividido en funciones que serían llamadas desde la función *main* como dictan las buenas prácticas de C.

Funciones definidas por el usuario

Cuando necesitamos agrupar términos de un programa podemos crear funciones que cumplan con la tarea.

```
tipo_retorno funcion_creada_usuario(lista de parametros)
{
    sentencias
    return;
}
```

Nombraremos a la función según lo que vaya a desempeñar.

Para utilizar una función en cualquier parte del código del programa se debe llamar a la función. Esta acción se denomina *llamada a función* y se realiza escribiendo el nombre de la

función incluyendo entre paréntesis los valores que debe recibir y finalizando en punto y coma (;).

Si el tipo_retorno es *void* la función no devuelve ningún valor, por lo tanto, estará a nuestra elección aprovecharlo o no.

```
#include <stdio.h>

tipo1 funcion_creada_usuario(tipo2 var1, tipo3 var2)      //funcion
{
    ...
}

int main()
{
    int num1, int num2;

    ...

    funcion_creada_usuario(num1, num2);                      //llamada a funcion

    ...

    return 0;
}
```

Creación de un programa

Para crear un programa estándar de C se pueden seguir los siguientes pasos:

- Definir el programa.
- Definir directivas del preprocesador.
- Definir declaraciones globales.
- Crear main().
- Crear el cuerpo del programa.
- Crear funciones definidas por el usuario.
- Compilar, enlazar, ejecutar y comprobar el programa.
- Utilizar comentarios.

Depuración de un programa

El concepto de *depuración* es el proceso de encontrar y corregir errores. En la mayoría de ocasiones lo más complicado de la depuración es detectar el error.

El compilador es el encargado de detectar los errores, pero en ocasiones son difíciles de interpretar. Una vez más la experiencia es la mejor aliada. Los errores de programación se pueden agrupar en tres tipos: sintaxis, lógicos y de regresión.

- **Errores de sintaxis:** se produce cuando el programa viola las reglas gramaticales del lenguaje C, por ejemplo, escritura incorrecta, falta de paréntesis o llaves, omisión de signos de puntuación, mala escritura de palabras reservadas o funciones, ... Estos errores son detectados por el compilador en la fase de traducción del código fuente a lenguaje máquina y se consideran los más fáciles de detectar.
- **Errores lógicos:** representan errores del programador en el diseño del algoritmo, es decir, cuando se está programando se confunde por ejemplo los signos matemáticos. Estos errores son muy difíciles de detectar y más aún cuando el programa es extenso.
- **Errores de regresión:** son aquellos que se crean accidentalmente cuando se intenta corregir un error lógico, así que como se suele decir en programación “un error se ha introducido, probablemente por el último código modificado”.

Mensajes de error del compilador

Los mensajes de error según los muestra el compilador suelen ser del tipo sintaxis (es decir, relacionado con errores de escritura que no siguen las reglas de C) y hay tres diferentes:

- **Errores fatales:** son errores que raramente se producen, suelen estar relacionados con trabas en el compilador. Cuando ocurre un error fatal la compilación se detiene inmediatamente. Se requerirá de acciones como volver a compilar o, incluso reiniciar el compilador.
- **Errores de sintaxis:** los comprenden los errores de sintaxis, los errores de línea de órdenes y los errores de acceso a memoria o disco. El compilador se detendrá sin compilar el programa.
- **Advertencias (warnings):** las advertencias no interrumpen la compilación, solamente indican que algo parece no estar puesto de forma correcta.

Errores en tiempo de ejecución

Los errores de ejecución son aquellos que son detectados por el programa en tiempo de ejecución y que en ocasiones permiten la terminación del programa.

Se producen por realizar una operación ilegal (como $x/0$, $\sqrt{-x}$) , por manipular datos no válidos, por desbordamiento en un tipo de dato (int, double, ...), por intentar leer un archivo no creado, por asignar memoria dinámica no disponible, entre otros.

Los elementos de un programa C

Los tokens son los elementos léxicos de un programa en C y existen 5 clases: identificadores, palabras reservadas, literales, operadores y otros separadores.

- **Identificadores:** un identificador es una secuencia de caracteres, letras, dígitos y subrayado (_) que debe comenzar por letra (en algún compilador se admite el subrayado). Se distinguen mayúsculas y minúsculas. Los identificadores no pueden ser palabras reservadas (if, switch, else, ...)

- **Palabras reservadas:** una palabra reservada es una característica del lenguaje C asociada con algún significado especial. Una palabra reservada no se puede utilizar como nombre de identificador o función.

Palabras reservadas de C		
auto	float	signed
break	for	sizeof
case	goto	static
char	if	struct
const	inline (since C99)	switch
continue	int	typedef
default	long	union
do	register	unsigned
double	restrict (since C99)	void
else	return	volatile
enum	short	while
extern		

Ilustración 26.- Palabras reservadas de C

- **Operadores y separadores:** los diferentes operadores y separadores de C son:

Signos de puntuación y separadores de C											
!	%	^	&	*	O	-	+	=	{	}	_
[]	\	;	'	:	<	>	?	,	.	/

Ilustración 27.- Signos de puntuación y separadores de C

Tipos de datos en C

Esencialmente, casi todos los tipos de datos de C son números. C contiene pocos tipos de datos, pero tiene la capacidad para que el usuario cree sus propios tipos de datos, es decir, contiene los tipos básicos para la creación de otros. Los tres tipos básicos son: enteros, reales (de coma flotante) y caracteres.

Tipos de datos en C		
Tipo	Tamaño en bytes	Rango Mínimo a Máximo
int	4	-2147483648 a 2147483637
unsigned int	4	0 a 4294967295
short	2	-32768 a 32767
unsigned short	2	0 a 65535
long	4	-2147483648 a 2147483637
unsigned long	4	0 a 4294967295
float	4	3.4e-38 a 3.4e+38
double	8	1.7e-308 a 1.7e+308
long double	8	Depende del compilador
char	1	-128 a 127
unsigned char	1	0 a 255

Ilustración 28.- Tipos de datos en C

Nota: los tipos int, char y double tienen a su vez modificadores de tipos de datos como short, long, signed y unsigned para permitir un uso más concreto de los tipos de datos.

Numeración en base octal y hexadecimal

El lenguaje C puede incluir numeración en base diferentes como base decimal, octal y hexadecimal. De hecho, si se desarrollara software o hardware para computadora sería útil utilizar las bases octal o hexadecimal.

Para indicar en C que un número está en *base octal* se pone un cero (0) delante del número.

Para indicar en C que un número está en *base hexadecimal* se pone un cero y una equis (0x) delante del número.

Decimal	Octal	Hexadecimal
Base 10	Base 8	Base 16
3	03	0x03
8	010	0x0A
16	020	0x10
23	027	0x17
65536	02000000	0x10000

Ilustración 29.- Expresión de números

Código de escape

Se denomina *código de escape* a una barra oblicua (\) seguida de uno de los caracteres específicos.

Código de escape	Significado	Código ASCII	
		Dec	Hex
\n	nueva línea	10	A
\r	retorno de carro	13	D
\t	tabulación horizontal	9	9
\v	tabulación vertical	11	B
\a	alerta (pitido sonoro)	7	7
\b	retroceso de espacio	8	8
\f	avance de página	12	C
\\\	barra inclinada inversa	92	5C
\'	comilla simple	39	27
\"	comilla doble	34	22
\?	signo de interrogación	63	3F
\ooo	número octal	Todos	Todos
\xhh	número hexadecimal	Todos	Todos

Ilustración 30.- Código de escape

Constantes carácter y constantes cadena

Un *constante carácter* consta de una solo pieza mientras que una *constante cadena* es la unión de varios caracteres. Para utilizar un solo carácter se encierra entre comillas simple ('...') mientras que para representar una cadena hay que colocar dos corchetes al final del nombre y encerrar el texto entre comillas dobles ("...").

Para utilizar varias líneas de código en una cadena se pondrá una barra invertida al final de cada línea (\).

```
char caracter = 'j';
char cadena[] = "esto es una cadena";

char varias_lineas = "esto es una cadena \
de varias lineas";
```

Nota: El compilador de C inserta el carácter nulo (\0) al final de una cadena cuando compila el programa, sin embargo, NO inserta el carácter nulo cuando compila un carácter, sino que lo trata como un entero. El carácter nulo sirve para que C sepa donde finaliza la cadena.

Constantes enumeradas

Las *constantes enumeradas* o *enumerados* (enum) permiten crear listas de elementos de un mismo tipo como por ejemplo los colores de un semáforo. También podremos crear variables de un tipo enumerado creado previamente.



```
//enumerado (enum)
enum semaforo {rojo, amarillo, verde};

//Declaracion e inicializacion de variable tipo semaforo
enum semaforo Sem1 = rojo;

Sem1 = verde;           //ahora Sem1 es verde
Sem1 = amarillo;        //por ultimo Sem1 es amarillo
```

El compilador de C asigna valores al enumerado empezando por cero. Así rojo=0, amarillo=1 y verde=2, más tarde podremos hacer referencia a dichos valores por el nombre o por el número correspondiente.

Otras: const y volatile

El cualificador *const* hace que la variable creada no sea modificable por el código del programa. En la mayoría de los casos se recomienda su utilización sobre *#define* porque el código generado es más eficiente, por ejemplo, se puede especificar el tipo de dato, entre otros.

```
//Formato const
const tipo nom_var = valor;

//Declaracion e inicializacion de variable const
const int meses_anyo = 12; //meses_anyo siempre sera 12
```

El cualificador *volatile* hace que la variable pueda ser modificada por un hardware o software externo a C, además de por el programa. El cualificador *volatile* está destinado a programación muy avanzada de C. Este cualificador es muy utilizado para registro de periféricos externos.

```
//Formato volatile
volatile tipo nom_var = valor;

//Declaracion e inicializacion de variable volatile
volatile int registro_ext = 1056; //modificable por el codigo
// y de forma externa
```

Declaración e inicialización de variables

Una *declaración de una variable* es una sentencia que da información al compilador sobre el nombre y tipo de una variable. El compilador reservará la cantidad de memoria necesaria para almacenar los valores de dicha variable. Para utilizar una variable debe haber sido creada previamente.

```
long num_exento;    //numero entero
float num_decimal; //numero en coma flotante
double acumulacion; //numero en coma flotante
char caracter;     //caracter
```

Nota: En una declaración nunca se asigna un valor, es desempeño de una *inicialización*.

Una *inicialización de una variable* es una sentencia que da información al compilador sobre el nombre y tipo de una variable, además de darle un valor inicial.

```
long num_extenso = 12345643;
float num_decimal = 3.456;
double acumulacion = 12345335,6874637;
char caracter = "v";
```

Alcance de una variable

El **ámbito o alcance** de una variable se puede definir como la vida útil de la variable en el bloque de código donde está ubicada. Si la variable estuviera ubicada en el bloque principal (main) podría utilizarse en todo el bloque principal además de en todos los sub-bloques que incluyese (for, if, while, ...), sin embargo, una variable de una función externa no podría utilizarse. De este modo existen tres tipos de variables según su ámbito:

- Las **variables locales** son variables definidas dentro de una función, y son solamente visibles en esa función específica. Por lo tanto, una variable local puede llamarse del mismo modo en funciones distintas. Las variables locales predeterminadamente llevan el especificador *auto*, es decir, se crean en la función y se destruyen “automáticamente” cuando termina. De forma manual se puede incluir el especificador *static* para impedir que se destruyan y mantengan su valor EN LA FUNCIÓN.
 - Las variables *de ámbito de bloque* son un subtipo de variable local. Estas variables son visibles en el bloque {} donde son creadas, por ejemplo, si están ubicadas en un if, if-else, for, ... Al igual que las variables locales, una vez finalizado el bloque dejan de existir.
- Las **variables globales** son las variables definidas fuera de la función (donde están ubicados los #include y #define) y son visibles por todas las partes del programa. Las variables globales predeterminadamente contienen el especificador *static*.
- Las **variables dinámicas** pueden ser de ámbito local o de ámbito global, según donde sean creadas. La diferencia de una variable normal y una dinámica es que la dinámica se puede crear y cambiar su tamaño durante la ejecución del programa, al contrario que una variable normal que debe ser creada antes de ejecutar el programa. Se gestionan mediante funciones especiales del lenguaje.

Entradas y salidas

Los programas de C interactúan con el exterior mediante datos de entrada y salida.

Los *datos de entrada* serán introducidos por teclado o por archivo (.txt, .xls, ...).

Los *datos de salida* saldrán por pantalla, impresora o archivos (.txt, .xls, ...).

`printf()`

`printf()` es la función encargada de la salida por pantalla. Su función, aparte de mostrar texto por pantalla, es dar formato a los datos de salida que incluya.

```
//Formato printf()
printf("cadena_de_control", dato1, dato2, ...);
```

El formato de *printf()* se divide en dos partes: la cadena de control y los datos a mostrar (dato1, dato2, ...). La cadena de control incluye el *cuerpo del mensaje* ("...") y el *código de formato* para el tipo de dato (%d, %c, ...).

Códigos de formato para el tipo de dato	
%d	Entero con signo, en base decimal
%i	Entero con signo, en base decimal
%o	Entero en base octal
%x	Entero en base hexadecimal. Dígitos a-f
%X	Entero en base hexadecimal. Dígitos A-F
%u	Entero sin signo, en base decimal
%c	EL carácter indicado con su código ASCII
%e	Número real con signo en formato científico
%E	Igual a %e. Representado con 'E'
%g	Número real con signo a elegir formato más corto entre e ó f
%G	Número real con signo a elegir formato más corto entre E ó f
%f	Número real con signo
%s	Cadena de caracteres
%p	Dirección de memoria
%Lf	Número real con signo (long double)

Ilustración 31.- Código de formato para el tipo de dato *printf*

```
//Declaraciones e inicializaciones
int i = 11, j = 12;
char c = 'A';
float n = 40.87432;

printf("[%x %6d %c %.3f]", i, j, c, n); //Impr -> [b      12 A 40.874]
```

Ejemplo 2.- Formas de representar datos en *printf()*

El primer código de formato de la cadena de control (%x) imprime 'b' que es 11 en hexadecimal, el segundo (%6d) imprime '12' desplazado 6 huecos, el tercero imprime el carácter 'A' y el cuarto (%.3f) imprime el numero tipo float con tres dígitos decimales (.3). Adicionalmente si en el segundo caso pusiéramos "%-6d" los huecos se pondrán a la derecha de '12'.

scanf()

scanf() tiene la función de leer y dar formato a los datos introducidos por teclado. Para leer un dato y guardar lo correctamente en una variable, dicha variable debe estar direccionada o referenciada. El operador **dirección** de C es & y debe ir precedido al nombre de la variable.



```
//Formato scanf()
scanf("cadena_de_control", dato1, dato2, ...);

//Ejemplo
int i;
float j;
char c;

printf("Introduzca un entero, un numero \
con decimales y un caracter: ");
scanf("%d %f %c", &i, &j, &c); //dirección (&)
printf("Valores: %d, %f, %c", i , j, c);
```

Ejemplo 3.- Utilización de scanf()

Introduzca un entero, un numero con decimales y un caracter: 56 2.369 k
 Valores: 56, 2.369000, k

Salida 1.- Salida del Ejemplo 5

Códigos de formato para el tipo de dato	
%d	Entero con signo, en base decimal
%i	Entero con signo, en base decimal
%o	Entero en base octal
%x	Entero en base hexadecimal. Dígitos a-f
%X	Entero en base hexadecimal. Dígitos A-F
%u	Entero sin signo, en base decimal
%c	EL carácter indicado con su código ASCII
%e	Número real con signo en formato científico
%f	Número real con signo
%s	Cadena de caracteres
%p	Dirección de memoria
%Lf	Número real con signo (long double)

Ilustración 32.- Código de formato para el tipo de dato scanf

puts()

El cometido de *puts()* es solamente imprimir cadenas de caracteres. Funciona de modo parecido a *printf()*, concretamente imprime la cadena y añade un fin de línea (\n).



```
//Formato puts()
puts(cadena);

//Ejemplo 1. Imprime por pantalla el texto entre comillas
puts("Esta es la funcion puts");

//Ejemplo 2. Imprime por pantalla A, el contenido de c
char c = 'A';
puts(c);

//Ejemplo 3.
//Imprime Titulo dejando una tabulacion a la izquierda
puts("\tTitulo");
```

gets()

La función de `gets()` es la de obtener una **cadena de caracteres** con una longitud deseada desde el teclado. Coge los caracteres incluidos espacios hasta leer el carácter nueva línea (`\n`), el cual elimina, y los guarda en una variable, al contrario que `scanf()` que solo coge caracteres antes del primer espacio o del carácter nueva línea. Al final de la cadena obtenida, `gets()` inserta un final de línea (`\0`).

```
//Formato gets()
gets(cadena);

//Ejemplo
char linea[81]; //cadena de max 80 caract mas el caracter nulo (\0)

printf("Nombre y direccion: ");
gets(linea); //obtiene caracteres hasta encontrar (\n)

puts(linea);
```

Aviso: `scanf()` coge caracteres hasta encontrar el carácter espacio o nueva línea, pero deja uno de los dos caracteres en el flujo de entrada. Si en el código posterior hay una función `gets()`, esta cogerá uno de los caracteres provocando anomalías de ejecución. Si fuese el espacio la cadena obtenida llevará un espacio al comienzo, pero ¡OJO! si fuese el carácter nueva línea, la cadena no contendrá NADA.



Ejercicio 1

```
1  /*Ejercicio 1*/
2
3  #include <stdio.h>
4
5  void main()
6  {
7      long numero;
8      char letra;
9      puts("\tPrograma que imprime un DNI por pantalla");
10
11     printf("Proporcioname una letra: ");
12     scanf("%c", &letra);
13     printf("Proporcioname un numero de 8 digitos: ");
14     scanf("%d", &numero);
15
16     printf("\nDNI: %d-%c", numero, letra);
17 }
```

Ejercicio 1.- Programa que obtiene e imprime un nombre y una dirección

Capítulo 4. Operadores y expresiones

Introducción

Operadores de C			
Nivel	Operadores	Descripción	Asociatividad
1	() [] -> .	Acceso a un elemento de un vector y paréntesis	Izquierdas
2	+ - ! ~	Signo pos y nega, negación lógica, negación bit a bit	Derechas
	* &	Acceso a un elemento: puntero y dirección	
	++ --	Incremento y decremento (pre y post)	
	(cast) sizeof	Cambio de tipo y tamaño de un elemento	
3	* / %	Producto, división y resto	Izquierdas
4	+ -	Suma y resta	Izquierdas
5	>> <<	Desplazamiento	Izquierdas
6	< <= > = >	Comparaciones de superioridad e inferioridad	Izquierdas
7	== !=	Comparaciones de igualdad	Izquierdas
8	&	'And' bit a bit binario	Izquierdas
9	^	'Exclusive-Or' binario	Izquierdas
10		'Or' bit a bit binario	Izquierdas
11	&&	'And' lógico	Izquierdas
12		'Or' lógico	Izquierdas
13	?:	Condicional	Derechas
14	= *= /= %= += -=	Asignaciones de operaciones aritméticas	Derechas
	>>= <<= &= ^= =	Asignaciones de operaciones bit a bit	
	,	Coma	Izquierdas

Ilustración 33.- Operadores de C

La *Ilustración* contiene todos los operadores de C ordenados por nivel de prioridad, siendo el nivel 1 es de máxima prioridad (el que se ejecuta antes) y el nivel 15 el de menor prioridad. La asociatividad indica por donde el compilador de C lee las expresiones de igual nivel de prioridad. Por ejemplo, la operación aritmética se ejecuta comenzando por la izquierda.

Operadores incremento y decremento

Los *operadores de incremento y decremento* hay que utilizarlos con cuidado porque dependiendo del contexto se pueden estar utilizando de forma incorrecta.

```
n = 3;
m = ++n;           //incrementa n en 1 y lo asigna a m. m=4
n = 4;
printf("n = %d", --n); //decrementa n en 1 y lo pasa a printf(). escribirá '3'

n = 3;
m = n++;          //asigna n a m, (m = 3) e incrementa n, (n = 4)
n = 4;
printf("n = %d", n--); //imprime '4' y decrementa n, (n = 3)
```

A modo resumen, si los operadores ++ o -- son prefijos de una variable, primero se incrementará o decrementará y después se asignará. Por el contrario, si los operadores ++ o -- son sufijos de una variable, primero se efectúa la asignación y después se incrementa o decremente.

Operadores de asignación e igualdad

Un error típico en programación es el de confundir los operadores asignación (=) e igualdad (==). El operador asignación (=) asigna el valor de la derecha a la izquierda. El operador igualdad (==) comprueba si el valor de la izquierda y el de la derecha son iguales, en caso de ser afirmativo devolverá '1' y en caso contrario '0'.

Nota: C no contiene tipos de datos booleanos. En su lugar, cualquier valor distinto de 0 será verdadero y el valor 0 indicará falso.

Operadores lógicos

Los *operadores lógicos* devuelven los valores 0 (falso) o 1 (verdadero). Las estructuras de C (if, for, while, ...) utilizan estos valores para verificar si se ejecutan.

Los operadores lógicos implementados en C son *not* (!), *and* (&&), y *or* (||). El operador lógico *not* (!) invierte el valor del operando (1->0, 0->1). El operador lógico 'and' (&&) solo es verdadero si los operandos a la izquierda y derecha de (&&) son verdaderos, en los demás casos será falso. El operador lógico 'or' (||) solo es falso si los dos operandos a la izquierda y a la derecha de (||) son falsos, en los demás casos será verdadero.

Operadores de manipulación de bits

Los *operadores de manipulación de bits* ejecutan operaciones lógicas a nivel de cada bit de un operando. Estas operaciones son comparables en eficiencia y velocidad al lenguaje ensamblador.

Estos operadores son & (Y), | (O), ^ (Exclusive-O), ~ (inversión), << (desplazamiento a la izquierda), >> (desplazamiento a la derecha), además de las combinaciones con el operador (=). Están comprendidos en la tabla de *Operadores de C* en los niveles de prioridad 5, 8, 9, 10 y algunos del 14.

Operador condicional

El operador condicional (?:) es un operador que requiere de tres operandos para su funcionamiento. Tiene el mismo funcionamiento que una sentencia if-else, es decir, si la condición impuesta es verdadera se ejecuta una sentencia y si es falsa se ejecuta otra.



```
//Formato operador condicional (?:)
condicion ? expresion_verdadera : expresion_falsa;

//Ejemplo
int volumen = 0;
char mensaje[20];

...

(volumen > 30) ? printf("Bajar vol") : printf("Subir vol");
//si volumen > 30 imprimir "Bajar vol", sino imprimir "Subir vol"
```

Conversiones de tipos

Las *conversiones de tipo* pueden ser hechas automáticamente por el compilador (*implícitas*) o específicamente por el programador (*explícitas*).

Las *conversiones implícitas* siguen la siguiente serie (int -> unsigned int -> long -> unsigned long -> float -> double).

Las *conversiones explícitas* se realizan de la siguiente forma **(tipo_conversion)valor**.

```
//Conversion implicita
int i = 5;
float t = 2;
i = i+t;      //float predomina sobre int, i se convertira a float

t = 3.0;
t = t/2;      //convierte 2 en tipo float y luego realiza la division

//Conversion explicita
int suma;
suma = (int)3.87 + (int)4.34;
//la conversion a int elimina los decimales. suma = 7
```

Ejercicio 2

El domingo de Pascua se calcula mediante las siguientes ecuaciones:

$$A = \text{año \% } 19$$

$$B = \text{año \% } 4$$

$$C = \text{año \% } 7$$

$$D = (19*A + 24) \% 30$$

$$E = (2*B + 4*C + 6*D + 5) \% 7$$

$$N = (22 + D + E)$$



La letra N indica el día del mes de **Marzo**, si N es menor o igual a 31, o el mes de **Abril**, si N es mayor que 31.

Construir un programa que calcule el domingo de Pascua para un año determinado. Utilizar el operador condicional (?:).

Nota: El símbolo % significa resto de la división entera en C. No confundir con /, que calcula el cociente de la división.

Aviso: La letra ñ no es válida en el lenguaje C.

```

1      //Programa que calcula el domingo de Pascua de un año determinado
2
3 #include <stdio.h>
4
5 int main()
6 {
7     int A,B,C,D,E,N,anyo;
8
9     printf("De que anyo quieres saber el dia del domingo de Pascua? ");
10    scanf("%d", &anyo);
11    //IMPORTANTE: operador dirección (&) para guardar valores
12
13    //ecuaciones de calculo
14    A = anyo % 19;
15    B = anyo % 4;
16    C = anyo % 7;
17    D = (19 * A + 24) %30;
18    E = (2 * B + 4 * C + 6 * D + 5) % 7;
19    N = (22 + D + E);
20
21    //operador ternario (?:) como sentencia if-else
22    N <= 31 ? printf("Domingo de Pascua: %d de Marzo de %d", N, anyo)
23        : printf("Domingo de Pascua: %d de Abril de %d", N - 31, anyo);
24    //N indica los días del mes de Marzo,
25    // pero si sobrepasa 31 estaremos en el mes de Abril
26 }
```

Ejercicio 2.- Programa que calcula el domingo de Pascua de un año determinado

Ejercicio 3

Realiza un programa para calcular la hipotenusa de un triángulo rectángulo a partir de sus catetos.

Nota: Puedes usar la función de cálculo de potencias **pow(base, potencia)** incluida en **<math.h>**, además de la del cálculo de raíces cuadradas **sqrt(x)**.

```

1      //Programa que calcula la hipotenusa de un triángulo rectángulo
2
3  #include <stdio.h>
4  #include <math.h>
5
6  int main()
7  {
8      float a, b, h, suma;
9
10     printf("          /*Calculo de la hipotenusa*/\n");
11
12     printf("Puedes darme los dos lados de un triangulo rectangulo\
13 separados por un espacio? ");
14     scanf("%f %f", &a, &b);
15
16     suma = pow(a, 2) + pow(b, 2);
17     h = sqrt(suma);
18
19     printf("La hipotenusa de un triangulo rectagulo con lados a [%f] \
20 y b [%f] es %f", a, b, h);
21 }
```

Ejercicio 3.- Programa que calcula la hipotenusa de un triángulo rectángulo

Capítulo 5. Estructuras de selección: sentencias if y switch

Definición estructura de control

Las *estructuras de control* tienen la capacidad de controlar el flujo de ejecución de un programa o función.

Cualquier código de C se ejecuta de arriba hacia abajo, pero mediante estructuras de control se puede cambiar el flujo de ejecución permitiendo seleccionar que código utilizar, o repetir una porción del código. Entonces, las estructuras de control se pueden clasificar en dos tipos: *selección* y *repetición*.

Entre las *sentencias de selección* se encuentran **if**, **if-else**, **if-else múltiples** y **switch**.

Entre las *sentencias de repetición* se encuentran **while**, **for** y **do-while**.

La sentencia if (una alternativa)

La sentencia *if* se trata de la estructura de control de selección principal. La sentencia *if* más sencilla es la que evalúa una condición y, si es verdadera ejecuta un bloque de sentencias. En caso contrario no se ejecutará nada.

```
//Formato simple if
if(condicion) sentencia;

//Formato compuesto if
if(condicion)
{
    sentencia1;
    sentencia2;
    ...
}

#include <stdio.h>

void main()
{
    int nota;

    printf("      /*Comprobar si se ha superado el examen*/\n");
    printf("Introduce una nota de 0 a 10\n");
    scanf("%d", &nota);

    if(nota >= 5) printf("Examen superado");
}
```



Diagrama de flujo 3.- La sentencia if

Nota: Múltiples sentencias deben ir encerradas entre llaves {...}.



if-else (dos alternativas)

Otra forma de evaluar la sentencia if es hacer una sentencia *if-else* que consiste en evaluar una condición, para ejecutar una sentencia (o grupo de sentencias) en caso verdadero y otra sentencia (o grupo de sentencias) en caso falso. La diferencia con la sentencia simple if es que en este caso sí hay una sentencia (o grupo de sentencias) que se ejecuta en caso de que no se cumpla la condición.

```
//Formato simple if-else
if(condicion) sentencia_v;
else sentencia_f;

//Formato compuesto if-else
if(condicion)
{
    sentencia_v1;
    sentencia_v2;
    ...
}
else
{
    sentencia_f1;
    sentencia_f2;
    ...
}

#include <stdio.h>

void main()
{
    float a, b;
    printf("      Cual es el mayor de dos numeros?\n");
    printf(" Introduce dos numeros\n");
    scanf("%f %f", &a, &b);

    if(a > b) printf("El primer numero es el mayor: %f", a);
    else printf("El segundo numero es el mayor: %f", b);
}
```

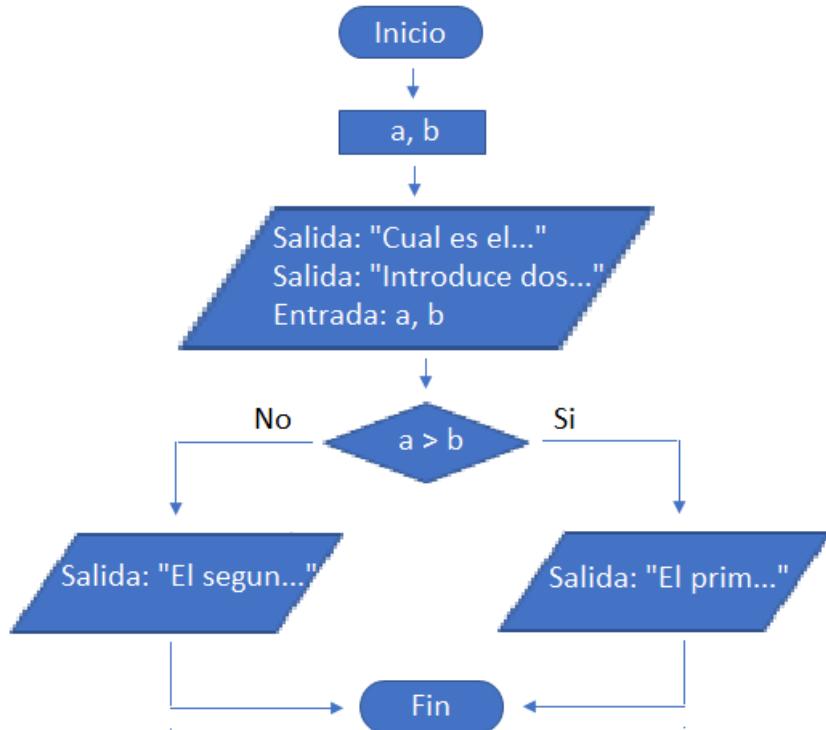


Diagrama de flujo 4.- if-else

if-else if-....-else (múltiples alternativas)

Las sentencias *if-else anidadas* (if-else if-....-else) son sentencias que contienen más de una condición. Cada nueva condición se evalúa en un nuevo *else if*, de forma que cada condición podrá tener sentencias verdaderas o falsas.

```
//Formato simple if-else anidado
if(condicion1) sentencia;
else if(condicion2) sentencia;
...
else sentencia;

//Formato compuesto if-else anidado
if(condicion1)
{
    sentencia1;
    sentencia2;
    ...
}
else if(condicion2)
{
    sentencia1;
    sentencia2;
    ...
}
...
else
{
    sentencia1;
    sentencia2;
    ...
}

#include <stdio.h>

void main()
{
    int nota;
    printf(" Que calificación le corresponde a una nota de examen?\n");
    printf(" Introduce una nota de examen [0, 10]\n");
    scanf("%d", &nota);

    if(nota < 5 && nota >= 0) printf("Insuficiente");
    else if(nota == 5) printf("Suficiente");
    else if(nota == 6) printf("Bien");
    else if(nota == 7 || nota == 8) printf("Notable");
    else if(nota == 9 || nota == 10) printf("Sobresaliente");
    else printf("El número introducido no está entre 0 y 10");
}
```

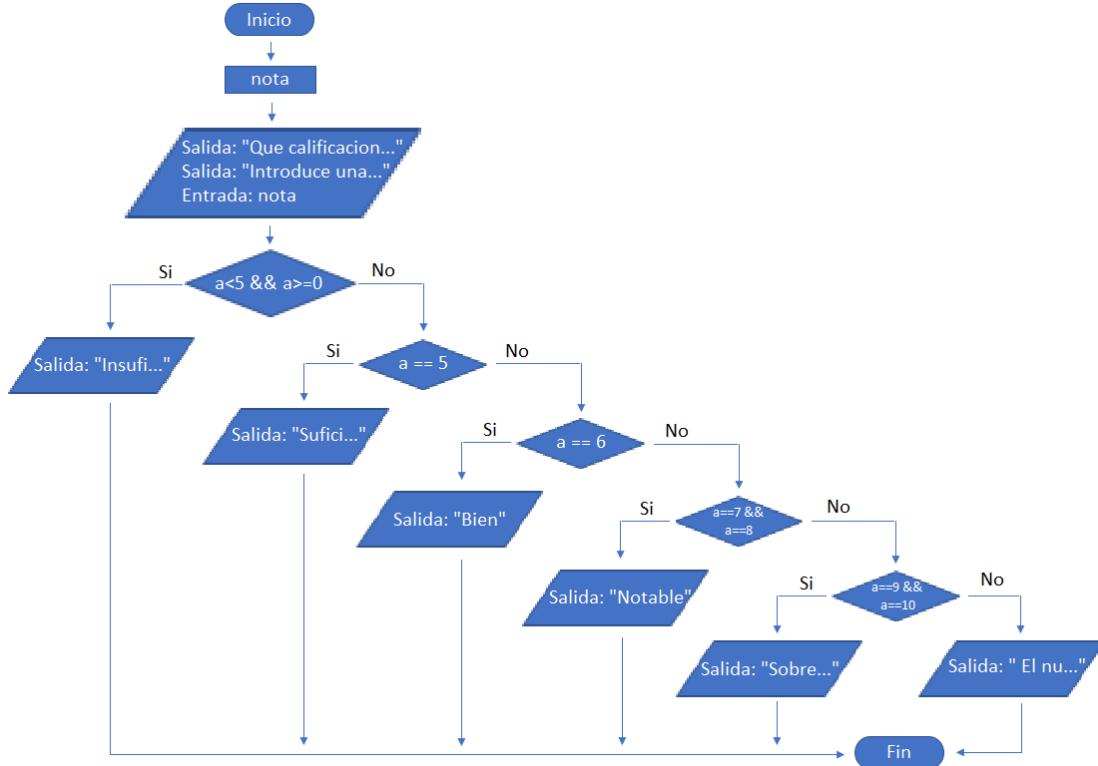


Diagrama de flujo 5.- if-else if-...-else

La sentencia switch (múltiples alternativas)

La sentencia *switch* es una sentencia de control de selección múltiple. Es utilizada cuando una sola variable (variable de control) sirve como condición para las múltiples salidas. La variable de control solo puede ser de tipo int o char.

```
//Formato switch
switch(variable_control)
{
    case etiqueta_1 :
        sentencias_1;
        break;
    case etiqueta_2 :
        sentencias_2;
        break;
    ...
    case etiqueta_x;
        sentencias_x;
        break;
    default :
        sentencias_def;
        ...
}
```



Cuando la variable _control cumple con alguna de las etiquetas, se ejecutan sus sentencias asociadas y termina (break) el switch. El caso por defecto se ejecutará si no se cumple ninguna etiqueta. Aunque el caso por defecto es opcional se recomienda su uso, su omisión podría crear errores lógicos difíciles de prever.

```
#include <stdio.h>

void main()
{
    char car;
    printf("    /*La letra introducida es una vocal?*/\n");
    printf("Introduce una letra\n");
    scanf("%c", &car);

    switch(car)
    {
        case 'a' : case 'A' :    //En caso de que un 'case' no tenga
        case 'e' : case 'E' :    // 'break' saltara de manera secuencial
        case 'i' : case 'I' :    // al siguiente 'case' hasta encontrar
        case 'o' : case 'O' :    // un 'break' o el caso por defecto
        case 'u' : case 'U' :
            printf("%c es una vocal", car);
            break;
        default :
            printf("%c no es una vocal", car);
    }
}
```

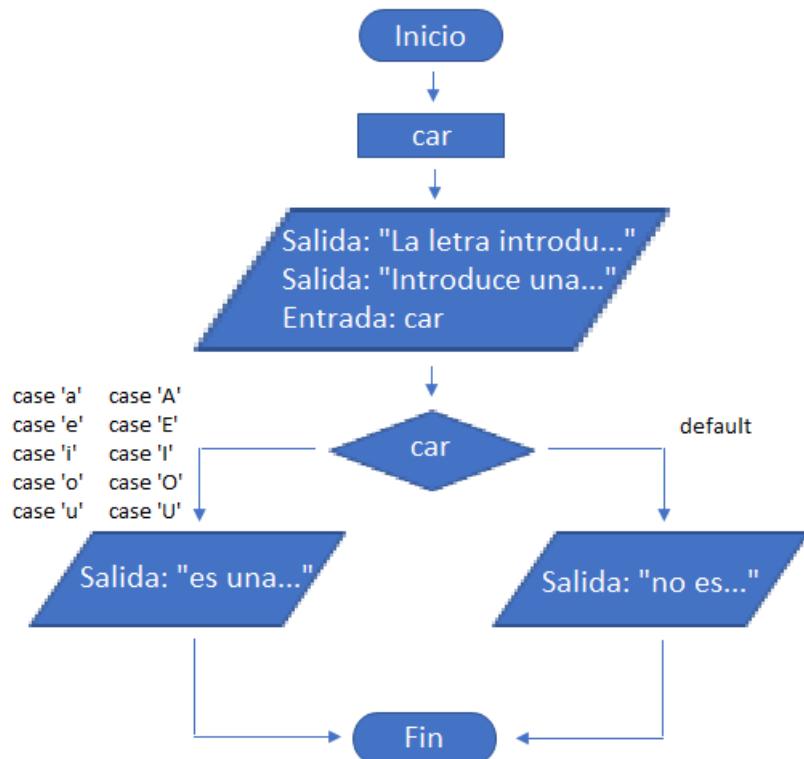


Diagrama de flujo 6.- switch

Ejercicio 4

Crea un programa que haga el funcionamiento de una calculadora de enteros, es decir, mediante la introducción por teclado de dos enteros más el signo (+, -, *, /, %) obtener un resultado. Se requiere un resultado entero.



```
1 #include <stdio.h>
2
3 void main()
4 {
5     int x, y;
6     char signo;
7
8     printf("      /*Calculadora de enteros simple*/\n");
9     printf(" Introduce el algoritmo a calcular [x ? y]\n");
10    scanf("%d %c %d", &x, &signo, &y);
11
12    switch(signo)
13    {
14        case '+': printf("Resultado: %d", x + y); break;
15        case '-': printf("Resultado: %d", x - y); break;
16        case '*': printf("Resultado: %d", x * y); break;
17        case '/': printf("Resultado: %d", x / y); break;
18        case '%': printf("Resultado: %d", x % y); break;
19        default : printf("El algoritmo introducido es incorrecto");
20    }
21 }
```

Ejercicio 4.- Calculadora de enteros simple

Ejercicio 5

Desarrolla un programa que calcule la edad de una persona.

Para ello se debe introducir (por teclado) el *día, mes y año de la fecha de nacimiento* y el *día, mes y año de la fecha actual*.

Por último, en caso de que sea menor de un año se debe mostrar en días y meses; en caso contrario mostrar en años.



```

1 #include <stdio.h>
2
3 void main()
4 {
5     int d_n, m_n, a_n, d_a, m_a, a_a;
6     int resta_d, resta_m, resta_a;
7     int opcion;
8
9     printf("          /*Calculo de la edad de una persona*/\n");
10    printf("Introduce la fecha de nacimiento [d m a]\n");
11    scanf("%d %d %d", &d_n, &m_n, &a_n);
12    printf("Introduce la fecha actual [d m a]\n");
13    scanf("%d %d %d", &d_a, &m_a, &a_a);
14
15    resta_a = a_a - a_n;
16    resta_m = m_a - m_n;
17    resta_d = d_a - d_n;
18
19    if(resta_a >= 0)           opcion = 1;
20    else if(resta_d >= 0 &&
21             resta_m >= 0 &&
22             resta_a >= 0)   opcion = 2;
23    else                      opcion = 3;
24
25    switch(opcion)
26    {
27        case 1 :
28            printf("La edad es: %d anyos", resta_a);
29            break;
30        case 2 :
31            printf("La edad es: %d dia(s) y %d mes(es)", resta_d, resta_m);
32            break;
33        case 3 : default :
34            printf("La fecha de nacimiento no puede superar a la actual");
35    }
36 }
```

Ejercicio 5.- Calculo de la edad de una persona

Capítulo 6. Estructuras repetitivas: bucles for, while y do-while

La sentencia while

La sentencia *while* es una estructura de repetición, es decir, es un bucle que se repite mientras se cumpla la condición impuesta. La particularidad de este bucle es que primero evalúa la condición y después realiza el cuerpo del bucle (sentencia o grupo de sentencias). Cada repetición se denomina iteración.



```
while(condicion_bucle) //formato while
{
    sentencia1;          //cuerpo
    sentencia2;          //cuerpo
    ...
    sentenciam;          //cuerpo
}
```

Normalmente la condición de bucle while depende de un número de ciclos predefinido, pero en ocasiones es más útil utilizarla para cortar el bucle a nuestro antojo como si de un *guardián* se tratase.

```
#include <stdio.h>

void main()
{
    int guardian =1, num = 0, sum = 0;

    while(guardian != -1)
    {
        printf("Introduce un numero para sumar\n");
        scanf("%d", &num);
        sum += num;
        printf("La suma total vale: %d\n", sum);

        printf("Quieres seguir sumando numeros? si [1] no [-1]\n");
        scanf("%d", &guardian);
    }
}
```

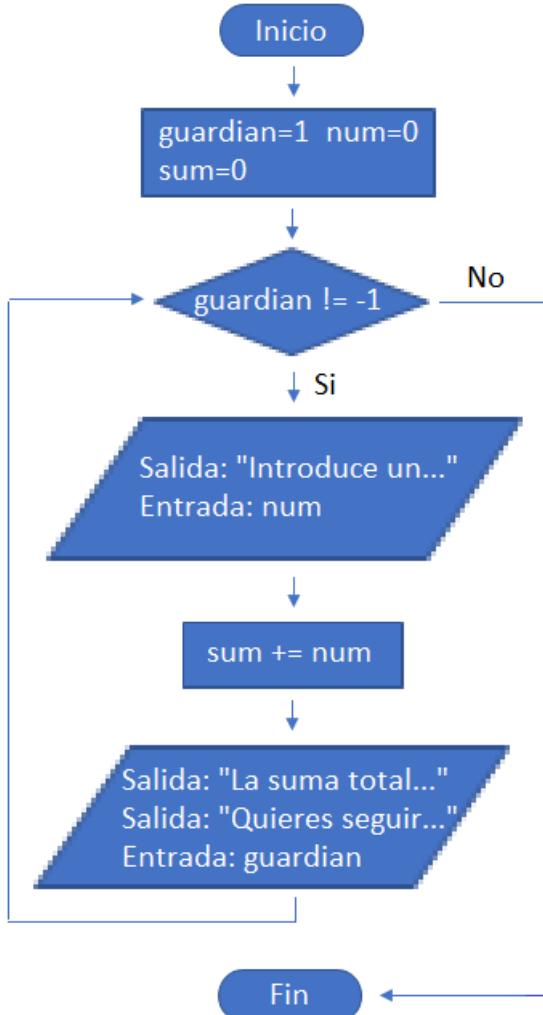


Diagrama de flujo 7.- while

Es posible la utilización de *break* para la ruptura de un bucle while o do-while. Las buenas prácticas de programación no recomiendan su uso ya que puede hacer difícil la comprensión y verificación del programa. Por suerte siempre habrá códigos alternativos que cumplirán con el mismo cometido.

La sentencia for

La sentencia *for* es una estructura de control de repetición. Es la forma más adecuada de implementar bucles controlados por un contador. Estos bucles están limitados a un número específico de repeticiones.



```
//Formato for
for(inicializacion; condicion; incremento) //cabecera
{
    sentencia_1;
    ...
    sentencia_x;
}
```

Las operaciones de control sobre el bucle for se sitúan en la cabecera, a diferencia del bucle while que solo sitúa la condición. Primero se inicializa la variable de control, segundo se comprueba si se cumple la condición, tercero se realizan todas las sentencias y cuarto incrementa la variable de control. El bucle for se ejecuta solo si la condición es verdadera.

Nota: La palabra reservada *break* puede romper el bucle for.

```
#include <stdio.h>

void main()
{
    int n, suma_p = 0, suma_i = 0;
    puts("Suma de los 10 primeros pares y los 10 primeros impares");
    for(n=1; n <= 10; n++)
    {
        suma_p += 2*n;
        suma_i += 2*n - 1;
    }
    printf("La suma de los 10 primeros numeros pares: %d\n", suma_p);
    printf("La suma de los 10 primeros numeros impares: %d", suma_i);
}
```

El diagrama de flujo de una estructura for es muy parecido al de una estructura while, salvo que inicializa una variable y aumenta un contador.

La sentencia do-while

La sentencia *do-while* es una sentencia de control de repetición. La diferencia con el bucle while es que primero se evalúa el cuerpo del bucle (sentencias) y después la condición, por lo tanto, el bucle se realizará al menos una vez. Esta pequeña diferencia hace que cada una sirva para resolver cuestiones diferentes.

```
do                  //formato do-while
{
    sentencia_1;   //cuerpo
    ...
    sentencia_x;  //cuerpo
}while(condicion);
```

El bucle do-while se repetirá solo si la condición es verdadera.

```
#include <stdio.h>

void main()
{
    int potencia = 1;
    puts("Visualiza las potencias de 2 hasta una que no sea superior a 1000");
    do
    {
        printf("%d ", potencia);
        potencia *= 2;
    }while(potencia < 1000);
}
```

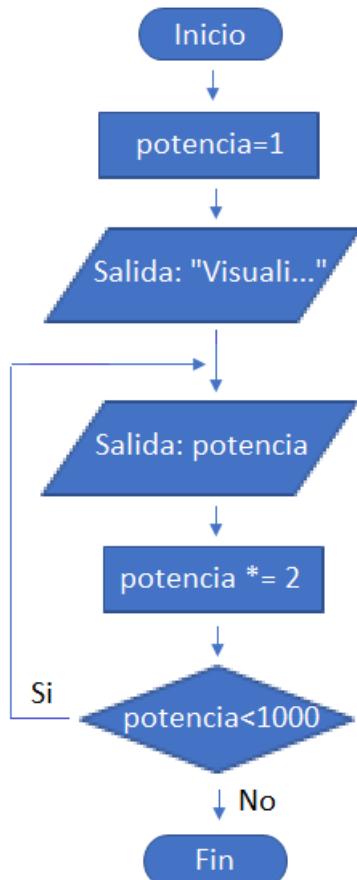


Diagrama de flujo 8.- do-while



Comparación de while, for y do-while

```
#include <stdio.h>

void main()
{
    int contador = 0;

    puts("/*Programa que ejecuta while, for y do-while\
con condiciones falsas para los tres bucles*/");

    printf("while: [");
    while(contador < 0)
    {
        printf("*");
        contador++;
    }

    printf("]\nfor: [");
    for(contador = 0; contador < 0; contador++)
    {
        printf("*");
    }

    contador = 0;
    printf("]\ndo-while: [");
    do
    {
        printf("*");
        contador++;
    }while(contador < 0);
    printf("]");
}

/*Programa que ejecuta while, for y do-while con condiciones falsas para los tres bucles*/
while: []
for: []
do-while: [*]
```

¿Cómo finalizar un bucle de control?

Los métodos más utilizados para finalizar un bucle de control (while, for y do-while) son:

- *Finalización por tamaño máximo predefinido*: Se utilizará un contador que irá aumentando con cada iteración, cuando el contador llegue al máximo, el bucle finalizará.
- *Finalización por pregunta al usuario*: La pregunta se realiza en cada iteración del bucle y si el usuario acepta la pregunta el bucle terminará.
- *Finalización por ‘guardián’*: Es muy parecido a la finalización por pregunta, con la diferencia de que el bucle finaliza si se introduce un valor distinto a los posibles.

- *Finalización por documento leído:* groso modo, cuando se leen archivos mediante bucle, al final del archivo hay una marca, *EOF (end of file)*, que indica la finalización del archivo y la ruptura del bucle.

Ejercicio 6

Los salarios de los trabajadores de una empresa van a ser aumentados de la siguiente forma:

Salario (€)	Aumento (%)
--------------------	--------------------

0 – 9000 20

9001 – 15000 10

15001 – 20000 5

Más de 20000 0

Entonces, desarrolla un programa que según el salario introducido haga el aumento correspondiente.

```

1 #include <stdio.h>
2
3 int main()
4 {
5     float salario;
6
7     printf("          /*Aumento de salario*/\n");
8     printf("Cual es el salario?\n");
9     scanf("%f", &salario);
10
11    if(salario > 0 && salario <= 9000)
12        printf("El salario aumenta un 20%: %.2f", salario * 1.20);
13    else if(salario > 9000 && salario <= 15000)
14        printf("El salario aumenta un 10%: %.2f", salario * 1.10);
15    else if(salario > 15000 && salario <= 20000)
16        printf("El salario aumenta un 5%: %.2f", salario * 1.05);
17    else if(salario > 20000)
18        printf("El salario no aumenta mas: %.2f", salario);
19    else
20        printf("El salario introducido no es correcto");
21 }
```

Ejercicio 6.- Aumento de salario

Ejercicio 7

Realiza un programa que calcule la media de **N** números introducidos por teclado. Adicionalmente debe decir cuál ha sido el número máximo y mínimo introducido.

```

1 #include <stdio.h>
2
3 int main()
4 {
5     float num, condicion;
6     float max = 0, min = 0, sumando = 0;
7     int contador = 0;
8
9     printf("      /*Calculo max, min y media de N numeros*/\n");
10    do
11    {
12        printf("Introduce el %d\xA7 numero\n", contador + 1); // \xA7 == °
13        scanf("%f", &num);
14
15        if(contador == 0) max = min = num; //la primera vez max=min=num
16        if(num > max) max = num;
17        if(num < min) min = num;
18        sumando += num;
19        contador++;
20
21        printf("Quieres introducir otro numero? si [1] no [-1] ");
22        scanf("%f", &condicion);
23    }while(condicion != -1);
24
25    printf("El numero maximo es: %.2f\n", max);
26    printf("El numero minimo es: %.2f\n", min);
27    printf("La media es: %.2f\n", sumando / contador);
28 }

```

Ejercicio 7.- Calculo del máximo, mínimo y media de N números

Ejercicio 8

El problema de Fibonacci sobre la cría de conejos dice que, partiendo de una pareja de conejos, estos tendrán dos crías cada mes (macho y hembra). Dichas crías serán fértiles al mes de nacer. Además, ningún conejo morirá.

Realiza un programa que diga cuantos meses serán necesarios para que hayan **N** conejos.



```

1 #include <stdio.h>
2
3 int main()
4 {
5     int N = 1; //Pareja de conejos inicial
6     int M = 0, meses;
7     float iteracion, meses_exactos;
8
9     puts("          /*Problema de Fibonacci sobre cria de conejos*/");
10    puts("Cada pareja de conejos tiene dos crias al mes. Cada cria \n");
11    es fertil al mes.\nCuantos meses se necesitan para tener M conejos si \n";
12    ninguno muere?\n");
13
14    printf("Dame el numero de conejos -> ");
15    scanf("%d", &M);
16    for(meses = 0; N < M; meses++)
17        N = 2 * N;
18    if(M < N)
19    {
20        iteracion = (float) (N-M) / (N-(N/2)); //ecuacion iteracion
21        meses_exactos = (1 - iteracion) + (meses - 1);
22    }
23    else
24        meses_exactos = meses;
25    printf("Dentro de aproximadamente %f mes(es) habra %d conejo(s)\n",
26           meses_exactos, M);
27

```

Ejercicio 8.- Cría de conejos

Ejercicio 9

Realiza un programa que muestre por pantalla el alfabeto en mayúsculas y en minúsculas. Adicionalmente se debe mostrar un triángulo alfabético por ejemplo en mayúsculas de la siguiente forma:

A

AB

ABC

ABCD...

ABC

AB

A

Nota: Puede resultar útil consultar la tabla de **código ASCII** en la web para saber la codificación de los caracteres *letra*.

```

1 #include <stdio.h>
2
3 int main()
4 {
5     puts("      /*Muestra el alfabeto en mayusulas y minusculas*/");
6
7     char alfabeto = 'A';      //'\x41'
8     while(alfabeto != 'Z')
9         printf("%c ", alfabeto++);
10
11    puts("");      //cambia a una nueva linea
12    alfabeto = '\x61';      //'a'
13    while(alfabeto != 'z')
14        printf("%c ", alfabeto++);
15    puts("\n");
16
17    puts("      %Triangulo 'alfabetico' en mayusulas%");
18
19    char ini;
20    for(alfabeto = 'A'; alfabeto <= 'Z'; alfabeto++)
21    {
22        for(ini = 'A'; ini <= alfabeto; ini++)
23            printf("%c", ini);
24        puts("");
25    }
26    for(alfabeto = 'Z' - 1; alfabeto >= 'A'; alfabeto--)
27    {
28        for(ini = 'A'; ini <= alfabeto; ini++)
29            printf("%c", ini);
30        puts("");
31    }
32 }
```

Ejercicio 9.- Imprimir alfabeto

Capítulo 7. Funciones

Introducción

Una función es un mini-programa dentro de un programa. Cuando un programa se vuelve demasiado extenso, las funciones juegan un papel importantísimo en el acortamiento, aclaración y eficiencia.

Las funciones pueden ser llamadas por otra función, pero para conseguir un efecto alguna de esas llamadas debe ser hecha en la función principal (main). Estas funciones pueden ser utilizadas indefinidas veces. Así como la función principal (main), las funciones pueden hacer uso de cualquier sentencia, estructura, tipo de dato, algoritmos, ...

Pueden ser agrupadas en librerías (bibliotecas) y utilizadas en cualquier programa ahorrando tiempo de desarrollo.



C fue creado para cumplir con una programación modular (subprogramas del programa) que pasó a llamarse *programación estructurada* con la inclusión de estructuras de control. Las funciones están encargadas de la programación modular.

Estructura de una función

Las funciones en C no se pueden anidar, es decir, una función no se puede declarar dentro de otra función. Esto hace que C sea más coherente con los datos, evitando mezclas complejas de funciones que desembocarían en malas prácticas.

```
tipo_de_retorno nombreFuncion (lista_de_parametros)
{
    cuerpo_de_la_funcion;      //sentencias
    return expresion;          //"final de la funcion". dato de retorno
}

//Ejemplo
float Resta (float num1, float num2)
{
    float respuesta;
    respuesta = num1 - num2;
    return respuesta;
}
```

Una función se define especificando el tipo de dato de retorno, del nombre de la función, la lista de parámetros, del cuerpo de la función y del return.

Tipo de dato de retorno

El *tipo de dato de retorno* es el tipo de dato que devuelve la función una vez termina de ejecutarse. La palabra reservada *return* devuelve el valor que generalmente deberá coincidir con dicho tipo (un *return* de tipo “int” puede incluirse en un tipo dato de retorno “float” pero será considerado una mala práctica en C. El tipo de retorno *void* no devuelve nada y por lo tanto no necesita la sentencia *return*.)

Los tipos de datos de retorno de una función son casi todos los disponibles en C (*int*, *double*, *float*, *char*), además de punteros a los anteriores. Las funciones también soportan los tipos nuevo creado por el programador como estructuras (*struct*), uniones (*union*), enumerados (*enum*) y renombramiento (*typedef*). Cuando no se indica un dato de retorno, por defecto la función será tipo *int*.

```
//Ejemplos de tipos de datos de retorno
int cuadrado(...) {...}
long cubo(...) {...}
float acumulacion(...) {...}
char pasar_caracter_a_minuscula(...) {...}

//devolucion de un puntero a tipo...
int* func1 (...) {...}
double* func2 (...) {...}
char* func3 (...) {...}
float* func4 (...) {...}

//devolucion de un tipo de dato (struct Coche)
struct Coche CambiarMatricula(int num_matricula, ...) {...}

//funcion vacia
void imprimir (...) {...}

//devolucion del tipo por defecto (int)
SumarConejos(...) {...}
```

Nombre, lista de parámetros y cuerpo

El *nombre de la función* es la denominación que elegimos para la función. Por norma general deberá ser un identificativo de su desempeño.

El nombre de una función puede comenzar por letra o por subrayado (_) pero nunca por número. C distingue entre mayúsculas y minúsculas.

La *lista de parámetros* utiliza el siguiente formato (tipo1 parametro1, tipo2 parametro2, tipo3 parametro3, ...). Los parámetros son los datos obtenidos de otra parte del programa para la realización de la función. Hay funciones que no necesitan parámetros.

El *cuerpo de la función* comprende el código necesario para el correcto desempeño de la función.

El retorno

La palabra reservada *return* se encarga de devolver el valor final de la función. Una buena práctica sería devolver el valor del mismo tipo que el tipo de retorno. Se podrán devolver valores múltiples mediante punteros y estructuras (struct), pero nunca mediante arrays. Tampoco se pueden devolver funciones.

Una función tiene dos formas de terminación:

- Cuando encuentra una sentencia *return*. Una función puede tener varias sentencias *return*, cuando el programa encuentre la primera la función terminará.
- En caso de no encontrar la sentencia *return*, la función terminará cuando llegue a la llave de cierre “}”.



Llamada a una función

Las funciones, para poder ser ejecutadas, han de ser llamadas o invocadas. Una función puede ser llamada desde la función principal (main) o desde cualquier otra función.

El funcionamiento es el siguiente. Por ejemplo, en la ejecución del main se llega a un punto en el que hay una llamada a una función. La ejecución se desplaza a la función y la ejecuta.

Cuando llega a un return o a una llave de cierre "}" la función termina y devuelve la ejecución al main en el punto donde se ejecutó la función. La ejecución proseguirá con el resto del código.

```
#include <stdio.h>

int cal_coc(int num1, int num2)
{
    return num1 / num2;
}

int cal_res(int num1, int num2)
{
    return num1 % num2;
}

void main()
{
    puts(" /*Calculo cociente y resto de dos numeros enteros*/");
    puts("Introduce los dos numeros separados por un espacio");

    int dividendo, divisor, cociente, resto;
    scanf("%d %d", &dividendo, &divisor);

    while(dividendo < divisor)
    {
        puts("El dividendo debe ser mayor que el divisor");
        scanf("%d %d", &dividendo, &divisor);
    }

    cociente = cal_coc(dividendo, divisor);
    resto = cal_res(dividendo, divisor);

    printf("El cociente es %d y el resto es %d", cociente, resto);
}
```

Prototipo de una función

Generalmente las funciones son *definidas*, es decir, son creadas ocupando un espacio en la memoria. Pero hay otros casos en los que se *declaran* previamente para, al final del código, definirlas. La declaración de una función se denomina *prototipo* e indica al compilador de C que existe la función. El *prototipo* de una función contiene la cabecera de la función finalizando en punto y coma.

```
//Prototipo de función
tipo_retorno nombre_funcion(lista_parametros);

///Ejemplos
int cociente(int num1, int num2);
//los nombre de los lista_parametros en ocasiones
// se pueden omitir
int cociente(int, int);

struct Persona Buscar(char nombre[], int edad, char sexo);

//función sin parámetros
char* IntroducirNombreyApellidos(void);
```

Esta forma de escritura se utiliza por su minimalismo dando un vistazo más claro del funcionamiento del programa. En la mayoría de ocasiones solo es necesario indicar el tipo de los parámetros de la función, habiendo ocasiones en las que se deberá indicar el nombre del parámetro para una mejor comprensión del funcionamiento.

Paso de parámetros por valor

El paso de un parámetro a una función se puede realizar de dos modos: **paso por valor o paso por referencia**.

Paso por valor (o copia) significa que cuando C compila una función y la función llamada, el valor de los parámetros de la función llamada son copiados a la función. Por lo tanto, C no pasa las variables sino la copia de su valor.

Si se produjese un cambio en los valores pasados dentro de la función solo afectaría a dicha función y no al resto del código de llamada.

```
#include <stdio.h>

void funcion_local(int);

void main()
{
    int valor = 13;
    printf("Funcion main: El valor es %d\n", valor);
    funcion_local(valor);
    printf("Funcion main: El valor solo ha sido modificado \
en la funcion y por lo tanto es %d", valor);
}

void funcion_local(int num)
{
    printf("Funcion local: El valor es %d\n", num);
    num = 23;
    printf("Funcion local: El valor a sido modificado en la \
funcion y es %d\n", num);
}
```

```
Funcion main: El valor es 13
Funcion local: El valor es 13
Funcion local: El valor a sido modificado en la funcion y es 23
Funcion main: El valor solo ha sido modificado en la funcion y por lo tanto es 13
```

Nota: Por defecto el paso de parámetros a una función será por valor.

Paso de parámetros por referencia

El paso de un parámetro a una función se puede realizar de dos modos: **paso por valor o paso por referencia**.

Paso por referencia se utiliza cuando queremos que la variable pasada a la función sea modificada. Para ello no pasamos una copia del valor, sino que pasamos la variable en sí por así decirlo. C da la oportunidad de usar una referencia a la variable, es decir, dar una referencia a la dirección de memoria donde esta guardada la variable. De esta manera se consigue que la función pueda modificar la variable directamente y que cualquier cambio en ella afecte al ámbito de la función y al ámbito de llamada a la función.

```
#include <stdio.h>

void intercambio(int *, int*);

void main()
{
    int x = 3, y = 5;
    printf("El valor de x = %d y el de y = %d\n", x, y);
    intercambio(&x, &y);
    printf("Ahora el valor de x = %d y el de y = %d\n", x, y);
}

void intercambio(int* x, int *y)
{
    int cambio;
    printf("\n[La variable x con referencia de memoria 0x%lx \
es igual a %d]\n", x, *x);

    cambio = *y;
    *y = *x;
    *x = cambio;

    printf("[Ahora la variable x con referencia de memoria 0x%lx \
es igual a %d]\n\n", x, *x);
}
```

```
El valor de x = 3 y el de y = 5
[La variable x con referencia de memoria 0x61ff2c es igual a 3]
[Ahora la variable x con referencia de memoria 0x61ff2c es igual a 5]

Ahora el valor de x = 5 y el de y = 3
```

Como se indica en el ejemplo. En el prototipo de la función y en la función se indica que las variables son punteros a tipo int, mediante el operador asterisco (*), y en la llamada a la función se indica la referencia a la variable, mediante el operador referencia (&). En la función, al imprimir el valor de x obtenemos la referencia o dirección de memoria donde se guarda la variable (61ff2c) y al imprimir el valor de *x obtenemos el valor (3 o 5).

Resumen paso de parámetros		
Parámetros pasados...	Parámetros Función	Parámetros Llamada a función
por valor	int a	b
por referencia	int* a / int *a	&b

Ilustración 34.- Resumen paso de parámetros

Parámetros const

El especificador *const* en la declaración de una función da seguridad adicional al paso de parámetros. Hace que el valor de la variable sea constante en el interior de la función en todo momento y por lo tanto no pueda ser modificado.

Especificador const en los parámetros de una función			
Forma parámetro	Tipo	¿Cambia nom_var dentro de la función?	¿Modifica parámetros del exterior?
int nom_var	por valor	Si	No
int* nom_var	por referencia	Si	Si
const int nom_var	por valor	No	No
const int* nom_var	por referencia	No su contenido	No

Ilustración 35.-Especificador const en los parámetros de una función

Nota: Un parámetro con especificador *const* no permite modificar el valor de la variable pasada dentro de la función, sin embargo, si permite cambiar de dirección hacia otra variable. Normalmente esta característica no tiene mucho sentido ya que si se utiliza *const* es para mantener dicho valor inmodificable.

Macros con argumentos

Una *macro* es el nombre dado a un conjunto de instrucciones en el preprocesador (#). Las macros suelen ser definidas por el programador, aunque C posee algunas propias. En general, se desaconseja su uso. Las funciones de C están exactamente para esto. Además, aportan comprobación de tipos, lo cual añade una capa de seguridad a la hora de compilar.

Si se utilizan las macros con el mismo fin de una función, se conseguirá aumentar la velocidad de ejecución, pero también se aumentará el tamaño del programa. Al tratarse de una directiva (#) su ejecución es mucho más rápida. Pero a su vez la macro irá unida a la cantidad de usos, es decir, se copiará cada vez que se utilice en el programa.

```
#define nombre(parametros_sin_tipos) expresion      //formato Macro

//Ejemplo funcion
float area_circulo(float r)
{
    return (3.1415 * r * r);
}

//Similar Macro
#define area_circulo(r) (3.1415 * r * r)

//llamada a ambas
int resultado;
resultado = area_circulo(6);
```

Al definir una macro: entre el nombre y los parámetros no puede haber huecos, NO HAY COMPROBACIÓN DE TIPOS, hay que definir las operaciones con paréntesis para evitar problemas de prioridad, la expresión se puede definir en varias líneas encerrándola entre llaves {...} y utilizando (\) al final de cada línea, es posible utilizar sentencias if, if-else, ...

```
#include <stdio.h>

#define Cubo(l) (l*l*l)

void main()
{
    int num;

    puts("\t\tCalculo del cubo de un numero");
    printf("Dame un numero entero ");
    scanf("%d", &num);

    printf("El cubo de %d es %d", num, Cubo(num));
}
```

```
Calculo del cubo de un numero
Dame un numero entero 6
El cubo de 6 es 216
```

Variables estáticas

Las *variables estáticas* son variables que contienen el especificador *static* en la declaración de la variable. Estas variables son visibles en el punto donde han sido declaradas y al contrario que las variables locales o de ámbito de bloque no dejan de existir una vez termina su ámbito, eso sí, solo son visibles en dicho ámbito.

```
#include <stdio.h>

int fibonacci(void);

void main()
{
    int n = 20, i;
    printf("\t\tPrograma que imprime los 20 primeros numeros de Fibonacci\n");
    printf("Numeros: 0, 1 ");
    for(i = 2; i < n; i++)
        printf(", %d ", fibonacci());
}

int fibonacci()
{
    static int x = 0; //una variable static una vez es declarada
    static int y = 1; //perdura aunque termine su ambito, eso si,
    y = y + x; //solo pude ser utilizada en su ambito
    x = y - x;
    return y;
}
```

Numeros: 0, 1 , 1 , 2 , 3 , 5 , 8 , 13 , 21 , 34 , 55 , 89 , 144 , 233 , 377 ,
610 , 987 , 1597 , 2584 , 4181

Funciones de biblioteca (#include)

Las *funciones de biblioteca* o *funciones estándar* de C ofrecen soporte para las operaciones utilizadas con más frecuencia. Con la llamada a una función específica obtenemos el resultado deseado sin necesidad de escribir su código fuente.

Las funciones estándar están incluidas en los *archivos de cabecera estándar* de C (#include). Los archivos de cabecera son llamados entre ángulos y son los siguientes.

Archivos de cabecera estándar de C				
<assert.h>	<math.h>	<stdio.h>	<cctype.h>	<setjmp.h>
<string.h>	<errno.h>	<signal.h>	<float.h>	<limits.h>
<stddef.h>	<stdarg.h>	<time.h>		

Ilustración 36.- Archivos de cabecera estándar de C

La sentencia `#include` añade el archivo de cabecera correspondiente en el programa. Para llamar a una función específica se debe incluir su archivo de cabecera previamente.

Funciones de carácter

El archivo de cabecera `<cctype.h>` define funciones para la manipulación de caracteres. Estas funciones verifican si un carácter cumple con determinadas condiciones. Si se cumple la función devuelve verdadero (distinto de cero), sino devuelve falso (cero). Las funciones incluidas son:

Funciones de carácter <ctype.h>

Función	Verificación de
Funciones de comprobación alfabética y de dígitos	
int isalpha(char c)	Letra mayúscula o minúscula
int isupper(char c)	Letra mayúscula
int islower(char c)	Letra minúscula
int isdigit(char c)	Número decimal
int isxdigit(char c)	Número hexadecimal
int isalnum(char c)	Letra o número decimal. isalpha(c) isdigit(c)
Funciones de comprobación de caracteres especiales	
int cntrl(char c)	Carácter de control (ASCII 0 a 31)
int isgraph(char c)	Carácter imprimible (no de control) excepto espacio
int isprint(char c)	Carácter imprimible (ASCII 32 a 127) incluyendo espacio
int ispunct(char c)	Carácter de puntuación
int isspace(char c)	Carácter espacio ' ', nueva línea '\n', retorno de carro '\r', tabulación '\t', tabulación vertical '\v' o '\f'
Funciones de conversión	
int toupper(char c)	Convierte letra a mayúscula
int tolower(char c)	Convierte letra a minúscula

Ilustración 37.- Funciones de carácter

```
#include <stdio.h>
#include <ctype.h>

void main()
{
    char car;
    puts("\t\tComprobacion de caracteres alfabeticos");
    printf("Introduce un caracter alfabetico: ");
    scanf("%c", &car);

    while(isalpha(car))
    {
        puts("Caracter alfabetico!");
        printf("Introduce otro caracter alfabetico: ");
        scanf("%c", &car);
    }
    puts("\nCaracter no alfabetico. Fin del programa");
}
```

```
#include <stdio.h>
#include <ctype.h>

void main()
{
    char letra;
    puts("\t\tIntercambio de letra mayuscula<>minuscula");
    printf("Introduce una letra: ");
    scanf("%c", &letra);

    if(isupper(letra))
        printf("La letra minuscula de '%c' es '%c'", letra, tolower(letra));
    else if(islower(letra))
        printf("La letra mayuscula de '%c' es '%c'", letra, toupper(letra));
    else
        printf("No se ha introducido una letra");
}
```

Funciones numéricas

Las funciones numéricas disponibles en C son: matemáticas, trigonométricas, logarítmicas, exponenciales y aleatorias. La mayoría de las funciones numéricas están en el archivo de cabecera `<math.h>`. Los números aleatorios se encuentran `<stdlib.h>`.

Funciones numéricas

Función	Funcionamiento
Funciones matemáticas	
ceil(x)	Redondea al entero más cercano
fabs(x)	Devuelve el absoluto (positivo) de x
floor(x)	Redondea por defecto al entero más próximo
fmod(x, y)	Calculo del resto (en coma flotante) de x/y
pow(x, y)	Calcula x elevado a la potencia y
pow10(x)	Calcula 10 elevado a la potencia x
sqrt(x)	Calcula la raíz cuadrada de x
Funciones trigonométricas (radianes) <math.h>	
acos(x)	Devuelve el arco coseno de x
asin(x)	Devuelve el arco seno de x
atan(x)	Devuelve el arco tangente de x
atan2(x, y)	Devuelve el arco tangente de x/y
cos(x)	Calcula el coseno de x
sin(x)	Calcula el seno de x
tan(x)	Calcula la tangente de x
Funciones logarítmicas y exponenciales <math.h>	
exp(x)	Calcula la exponencial de x en base e
expl(x)	Igual a exp(x), pero x será tipo long double
log(x)	Calcula el logaritmo neperiano de x
logl(x)	Igual a log(x), pero x será tipo long double
log10(x)	Calcula el logaritmo decimal de x
log10l(x)	Igual a log10(x), pero x será de tipo long double
Funciones aleatorias <stdlib.h>	
rand(void)	Devuelve un numero entero aleatorio en cada llamada a rand(), pero mantiene la semilla (secuencia) en cada inicio del programa
srand(x)	Permite cambiar la secuencia aleatoria de la función rand(). x tiene que ser una variable que cambie con el tiempo, por ejemplo el valor devuelto por la función time(NULL)

Ilustración 38.- Funciones numéricas

Comentarios:

- Estas funciones siguen las reglas matemáticas básicas.
- Las funciones trigonométricas realizan los cálculos en radianes.
- Las semillas son secuencias de números aleatorios fijos, es decir, una semilla tiene siempre los mismos valores. Cada semilla tiene una secuencia aleatoria diferente y con la función srand() y una variable que cambien con el tiempo podemos conseguir secuencias diferentes.

Funciones de fecha y hora

Los microprocesadores contienen un reloj interno para controlar el microprocesador. Este mismo reloj es utilizado por C para calcular la fecha y la hora.



El archivo de cabecera <time.h> es el encargado de definir estructuras, macros y funciones para la manipulación de fechas y horas.

```
#include <stdio.h>
#include <time.h>

void main()
{
    float inicio, fin, resultado;

    puts("\t\tTiempo de ejecucion del calculo de 1 a 1.000.000 \
de uno en uno");

    inicio = clock();
    for(double i = 0; i <=1000000; i++);
    fin = clock();

    resultado = fin - inicio;
    printf("El tiempo en clicks es %f y en segundos es %f", resultado, resultado/CLK_TCK);

    puts("\n\n\tTiempo de ejecucion del calculo de 1 a 100.000.000 \
de uno en uno");

    inicio = clock();
    for(double i = 0; i <=100000000; i++);
    fin = clock();

    resultado = fin - inicio;
    printf("El tiempo en clicks es %f y en segundos es %f", resultado, resultado/CLK_TCK);
}
```

```
Tiempo de ejecucion del calculo de 1 a 1.000.000 de uno en uno
El tiempo en clicks es 6.000000 y en segundos es 0.006000
```

```
Tiempo de ejecucion del calculo de 1 a 100.000.000 de uno en uno
El tiempo en clicks es 753.000000 y en segundos es 0.753000
```

Para la utilización del archivo de cabecera <time.h> es muy aconsejable tener conocimiento de punteros y estructuras. Por lo tanto, el resto de funciones implementadas no se van a explicar. Se aconseja que una vez adquiridos dichos conocimientos, se busque información sobre la biblioteca <time.h> en la web.

Funciones de utilidad

Otras funciones útiles se encuentran en el archivo de cabecera <stdlib.h> y son:



Funciones de utilidad	
int abs(int n)	Devuelve el valor absoluto de n
long labs(long n)	Igual que abs(n), pero con tipos long
div_t div(int num, int den)	Guarda en struct div_t el cociente y resto de num/den
ldiv_t ldiv(long num, long den)	Igual que div(n, d), pero con tipos long

Ilustración 39.- Funciones de utilidad

Estas son las estructuras utilizadas en las funciones div(...) y ldiv(...):

```
typedef struct
{
    //miembros de la estructura div_t
    int quot; //cociente. Acceso a miembro -> (div_t)nombre.quot
    int rem; //resto. Acceso a miembro -> (div_t)nombre.rem
}div_t;

typedef struct
{
    long int quot; //cociente. (ldiv_t)nombre.quot
    long int rem; //resto. (ldiv_t)nombre.rem
}ldiv_t;
```

El procedimiento para el acceso a miembros de la estructura es crear una variable de la estructura correspondiente [div_t var_est;]. A continuación, llamar a la función deseada y guardarla en la variable [var_est = div(num, den);]. Por último, acceder a sus miembros [var_est.quot;].

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
    div_t CocRes;
    int num, den;

    puts("\t\tCalculo del cociente y resto con la funcion div(num, den)");
    printf("Que numeros enteros quieres dividir? x y\n");
    scanf("%d %d", &num, &den);

    CocRes = div(num, den);
    printf("%d/%d => c: %d, r: %d\n", num, den, CocRes.quot, CocRes.rem);
}
```

```

        Calculo del cociente y resto con la funcion div(num, den)
Que numeros enteros quieres dividir? x y
3 2
3/2 => c: 1, r: 1

```

Situación de variables locales y globales con especificadores static y auto

```

#include <stdio.h>      //archivo de cabecera

//variable global
int z;                  //Utilizable por este y otros archivos
//variable global estatica
static int j;            //Utilizable solo por este archivo

void funcion(void);     //prototipo de la funcion

void main()              //funcion principal (main)
{
    funcion();           //llamada a la funcion
    ...
}

void funcion()           //funcion
{
    //variable local automatica. Creada y destruida en la funcion
    int i;                //int i == auto int i
    //variable local estatica. Creada y mantenida en la funcion
    static int x;
    ...
}

```

En esta imagen se muestran los tipos de variables *locales* y *globales* y el funcionamiento de los especificadores *static* y *auto*.

Variables externas

Las variables o funciones externas (*extern*) son declaradas en un archivo y llamadas desde otro. En el archivo de llamada se debe colocar el especificador *extern* antes del tipo de la variable y debe haber una variable con el mismo nombre en el otro archivo. Para funciones, se llama a su prototipo con el especificador *extern*, aunque por defecto C toma las funciones como *extern*.

Siempre es posible evitar el uso de este tipo de variable en C mediante un uso eficiente del resto de estructuras del lenguaje. Por tanto, no debe utilizar variables de tipo *extern*.

```
/*Código del archivo 1 -- main.c */
#include "var_fun.c"

extern int x; //variable ubicada en otro archivo

extern void funcion(void); //función ubicada en otro archivo

void main()
{
    ...
}

/*Código del archivo 2 -- var_fun.c */
int x;
static int v; //Utilizable solo por este archivo

void funcion()
{
    ...
}

static void otraFuncion() //Utilizable solo por este archivo
{
    ...
}
```

Nota: una variable o función global con especificador *static* no puede ser llamada (*extern*) por otro archivo.

Grandes programas. ¿Cómo afectan los especificadores static, extern y auto?

Los programas grandes requieren una gran gestión y mantenimiento, así que se desarrollan en torno a la programación modular siendo la opción para combatir con grandes extensiones. Cuando tenemos varios módulos bien delimitados y nos encontramos con un error, podemos aislar el fallo rápidamente. Por otra parte, el tiempo de compilación se ve reducido ya que solo se compilan los módulos modificados. Un programa modular tendrá ventaja en el mantenimiento por su sencilla interpretación.

La forma de compilación de grandes programas comienza compilando los módulos (módulos fuente) produciendo módulos objetos. El conjunto de módulos objeto son enlazados por el enlazador (link), produciendo el programa ejecutable.

Para conseguir una delimitación óptima entre los módulos:

- **Funciones:** las funciones por defecto llevan implícito el especificador *extern*, pero también se puede incluir manualmente el especificador *static*. Por lo tanto, se puede dejar la función pública a otros módulos o por el contrario privada para el propio. Las buenas prácticas de programación aconsejan la utilización de funciones privadas

(static) por posibles conflictos de llamada a funciones con el mismo nombre y por aumentar la claridad del programa.

- **Variables:** del mismo modo que en funciones, las variables globales por defecto son publica, pero pueden hacerse privadas (static). Además, se aconseja que sean static en el mayor número posible. Aun así, lo que recomiendan las buenas prácticas de programación es hacer las variables locales para conseguir una mayor restricción y claridad de programación.

Resumiendo, el ámbito de actuación de una variable o una función va sujeto de la cantidad de problemas (errores, claridad, mantenimiento, erratas, ...). A mayor ámbito abarcado por cada variable o función mayor número de problemas.

Recursividad de funciones

Una función *recursiva* es aquella que se llama así misma de forma directa o indirecta. Una función *recursiva directa* se llamará así misma desde el propio cuerpo de la función, mientras que una función *recursiva indirecta* llamará a otra función la cual llamará de nuevo a la primera. Eso sí, un proceso recursivo debe contener condiciones de terminación para que no se repita indefinidamente.

El caso típico de función recursiva directa es la resolución del número factorial.

```
#include <stdio.h>

long int factorial(long int);

int main()
{
    long int num, res;

    puts("\t\t/*Resolucion del numero factorial con funciones recursivas*/");
    printf("De que numero quieres saber su faltorial: ");
    scanf("%d", &num);

    res = factorial(num);

    printf("%d! = %d\n", num, res);
}

long int factorial(long int n)
{
    if(n > 1)
        return (n * factorial(n-1)); //recursividad directa
    return 1;
}
```

Ejercicio 11

Diseñe un programa que invierta una cadena de caracteres. Para ello cree una función que se encargue de invertir cualquier cadena de caracteres dada.

Nota: El carácter nulo `\0` indica el final de una cadena.

La función `gets()` es útil para la obtención de varias palabras. Dicha función obtiene caracteres introducidos por teclado hasta encontrar el carácter nueva línea `\n`, además finaliza la cadena con `\0`.

```

1  #include <stdio.h>
2
3  void imprimir_inversa(char* );
4
5  int main()
6  {
7      char cadena[50];
8
9      puts("\t\t/*Inversion de una cadena de caracteres*/");
10     printf("Introduce la cadena de caracteres -> ");
11     gets(cadena);    //coge varias palabras
12
13     imprimir_inversa(&cadena);          //variable referenciada
14 }
15
16 void imprimir_inversa(char *cadena) //variable referenciada
17 {
18     char cad_inv[50];
19     int tam, m, num_cad2 = 0;
20
21     //Calculo tamaño de la cadena
22     for(tam=0; cadena[tam] != '\0'; tam++);
23     //Copia inversa en otra cadena
24     for(m=tam-1; m>=0; m--)
25     {
26         cad_inv[num_cad2] = cadena[m];
27         num_cad2++;
28     }
29     cad_inv[num_cad2] = '\0';    // \0 -> fin de cadena
30
31     printf("La cadena inversa es [%s]", cad_inv);
32 }
```

Ejercicio 10.- Inversión de una cadena de caracteres

```

/*Inversion de una cadena de caracteres*/
Introduce la cadena de caracteres -> ISIDRO GOMEZ
La cadena inversa es [ZEMOG ORDISI]
```

Salida 2.- Salida del Ejercicio 11

Ejercicio 12

Realiza una función que calcule el seno, coseno y tangente de cualquier ángulo introducido por teclado.

Nota: Es útil el uso de las funciones ***sin(rad)***, ***cos(rad)***, ***tan(rad)*** incluidas en el archivo de cabecera ***<math.h>***. ¡OJO! Dichas funciones aceptan un ángulo en radianes en su argumento.

```

1 #include <stdio.h>
2 #include <math.h>
3 #define PI 3.14159265
4
5 void funciones_trigonometricas(float, double* );
6
7 void main()
8 {
9     float angulo, radian;
10    double soluciones[10];      //array
11
12    puts("\t\t/*Funciones trigonometricas*/");
13    printf("Ingresa el angulo(\xA7): ");
14    scanf("%f", &angulo);
15    radian = angulo*PI/180;
16
17    //paso de array por referencia
18    funciones_trigonometricas(radian, &soluciones);
19    printf("sen = %lf\n", soluciones[0]);
20    printf("cos = %lf\n", soluciones[1]);
21    printf("tan = %lf\n", soluciones[2]);
22 }
23
24 //soluciones y sol tienen la misma dirección de memoria
25 void funciones_trigonometricas(float rad, double *sol)
26 {
27     sol[0] = sin(rad);
28     sol[1] = cos(rad);
29     sol[2] = tan(rad);
30 }
```

Ejercicio 11.- Funciones trigonométricas

```

        /*Funciones trigonometricas*/
Ingresar el ángulo(º): 47
sen = 0.731354
cos = 0.681998
tan = 1.072369
```

Salida 3.- Salida del Ejercicio 12

Capítulo 8. Arrays (listas y tablas)

Introducción

Un *array* es una secuencia de datos del mismo tipo. Cada dato es un *elemento* del array, y se enumeran consecutivamente de cero en adelante (0, 1, 2, ...). El tipo de elemento puede ser de cualquier tipo de dato de C (char, int, float, ...), incluyendo estructuras definidas por el programador.

La declaración de un array en C debe contener el tipo, el nombre y el tamaño del array entre corchetes.

```
//Formato 1. declaracion de un array
tipo nombre[numero_elementos];
//Formato 2. declaracion e inicializacion
tipo nombre[] = {valor_0, valor_1, ..., valor_(n-1)};
//el numero de elementos es opcional en el Formato 2

int enteros[4]; //array de cuatro elementos de 0 a 3
char caracteres[10]; //array de caracteres
float punto_flotante[20];

char apellido1[] = {'S', 'a', 'n', 'c', 'h', 'e', 'z'};
//array de caracteres. apellido1 tiene 7 elementos
char apellido2[] = "Gomez";
//cadena de caracteres. apellido2 tendra 6 elementos,
//5 letras y el caracter nulo (\0)
//que indicara el final de la cadena de caracteres
float temperatura[] = {12.3, 12.1, 11.9};
//temperatura tendra 3 elementos de 0 a 2
printf("%f", temperatura[1]); // Imprimira 12.1
```

Si se declara e inicializa un array se puede no indicar el número de elementos. El compilador de C asignará el número de elementos automáticamente según la cantidad de valores dados.

Una diferencia clara entre un *array de caracteres* y una *cadena de caracteres* es que la cadena finalizará con el carácter nulo (\0) y el array no tiene por qué incluirlo. Para entender la situación es preciso decir que la cadena guarda un grupo de caracteres constante mientras que un array de caracteres tiene el objetivo de poder modificarlos.

Si al declarar un array, este es estático (static) o global, sus valores por defecto serán '0' para números (int, float, ...) o el carácter nulo (\0) para cadenas. De otro modo contendrán valores aleatorios.

Nota: C no comprueba si el elemento al que se quiere acceder está dentro de los límites del array produciendo fallos en el programa que pueden ser complicados de encontrar. Hay que prestar gran atención a este aspecto.

Aviso: Antes de la llegada del *estándar C99* (*estándar C90*), los arrays debían ser definidos en tiempo de compilación, es decir, el nombre del array y su tamaño se definía previamente antes

de ejecutar el programa. Tras la llegada del *estándar C99* se permitió que el tamaño del array ahora se pudiese decidir mientras se está ejecutando el programa (tiempo de ejecución).

```
void main()
{
    int tam;
    printf("Proporcioname el tamano del array: ");
    scanf("%d", &tam);

    int array[tam];
    ...
    return 0;
}
```

Arrays multidimensionales

Un *array multidimensional* implica más de una dimensión en su declaración. Cada dimensión se indica con una pareja de corchetes. En cada pareja de corchetes se indica el tamaño que tiene esa dimensión.

```
//Formato 1. declaracion de un array multidimensional
tipo nombre[num_filas][num_columnas]...[];
//Formato 2. declaracion e inicializacion
tipo nombre[][][m] = {{e00, e01, ..., e0(m-1)}, {e10, e11, ..., e1(m-1)}};
//al menos se debe indicar el numero de columnas (m)

int enteros[4][2]; //array de 4x2=8 elementos
char caracteres[10][3]; //array de caracteres
float punto_flotante[20][2][2]; //array de 20x2x2=80 elementos

char apellidos[][][7] = { {'S', 'a', 'n', 'c', 'h', 'e', 'z'},
| {'N', 'o', 'v', 'a'} };
//los huecos vacio de la segunda fila se rellenan con el car. nulo(\0)
printf("%c", apellidos[1][5]); //Imprimira un hueco correspondiente a (\0)
float temperatura[][][3] = { {12.3, 12.1, 11.9}, {11.5, 11.6, 11.7} };
//temperatura tendra 6 elementos
printf("%f", temperatura[1][0]); // Imprimira 11.5
```

En un array bidimensional el corchete más próximo al nombre es el número de filas y el otro es el número de columnas.

Los arrays multidimensionales se puede declarar e inicializar y hay que indicar el número de columnas como mínimo. Si se inicializan uno o más elementos, los huecos que no estén llenos los rellena el compilador con ceros (int, float, ...) o con el carácter nulo (char). Si solo declaramos en array este contendrá valores aleatorios.



```
#include <stdio.h>

int main()
{
    int a[3][5];

    int i, j;
    puts("Introduzca 15 numeros enteros, 5 por fila");
    for(i=0; i<3; i++)
    {
        printf("Fila %d: ", i);
        for(j=0; j<5; j++)
            |    scanf("%d", &a[i][j]);
    }

    for(i=0; i<3; i++)
    {
        for(j=0; j<5; j++)
            |    printf(" %d",a[i][j]);
        printf("\n");
    }
}
```

```
Introduzca 15 numeros enteros, 5 por fila
Fila 0: 6 5 8 4 2
Fila 1: 8 6 5 2 5
Fila 2: 9 8 5 1 2
6 5 8 4 2
8 6 5 2 5
9 8 5 1 2
```

Arrays como parámetro a funciones

Por defecto en C los arrays son pasados por referencia a las funciones. En la sintaxis no hay necesidad de indicar la referencia (&) en la llamada a la función ni el puntero (*) en la función. Hay que tener especial cuidado con esto, ya que una función que en principio se considera de lectura podría modificar los datos. Para suplir esta falta es muy aconsejable declarar la función con el parámetro array constante (const).



```
funcion(int array[]);  
  
int main()  
{  
    int a[2] = {12, 18};  
    printf("%d", a[0]);      //Imprimira 12  
    funcion(a);            //paso del array por referencia  
    printf("%d", a[0]);      //Imprimira 7  
    ...  
}  
  
funcion(int array[])
{
    array[0] = 7;
}
```

El prototipo debe ser escrito de forma completa, es decir, como la declaración de la función. Aunque no es siempre necesario pasar el tamaño del array, es aconsejable hacerlo siempre, indicando el tamaño de cada una de las dimensiones, pasándolas como otros parámetros adicionales a la función.

```
#include <stdio.h>

void leer(int a[][5]);
void visualizar(const int a[][5]);

int main()
{
    int a[3][5];

    leer(a);
    visualizar(a);
}

void leer(int a[][5])
{
    int i, j;
    puts("Introduzca 15 numeros enteros, 5 por fila");
    for(i=0; i<3; i++)
    {
        printf("Fila %d: ", i);
        for(j=0; j<5; j++)
            scanf("%d", &a[i][j]);
    }
}

void visualizar(const int a[][5])
{
    int i, j;
    for(i=0; i<3; i++)
    {
        for(j=0; j<5; j++)
            printf(" %d", a[i][j]);
        printf("\n");
    }
}
```

```
Introduzca 15 numeros enteros, 5 por fila
Fila 0: 5 2 4 6 9
Fila 1: 2 6 5 4 5
Fila 2: 2 5 5 9 3
5 2 4 6 9
2 6 5 4 5
2 5 5 9 3
```

Ordenación de listas

En la mayoría de ocasiones los datos son metidos uno tras otro. Para un buen manejo y claridad pueden ser ordenados.

El método utilizado para la ordenación es el *algoritmo de burbuja* que consiste en ir comparando dos números y subir el menor encima del mayor como si de una burbuja en un



líquido se tratase. Una vez se cumple la condición de que los números superiores son menores que los inferiores el algoritmo termina.

Si se quisiera ordenar la lista a la inversa se haría de la misma manera salvo que “flotarían los mayores”.



```
#include <stdio.h>
void imprimir(float c[], int);
void ordenar(float c[], int);
void intercambio(float*, float*);

void main()
{
    puts("\t\t/*Ordenacion de un array de numeros*/");
    float conjunto[7] = {12.3, 12.6, 13.1, 17.5, 17, 11.1, 13.27};
    printf("Secuencia: ");
    imprimir(conjunto, 7);
    ordenar(conjunto, 7);
    printf("Secuencia ordenada: ");
    imprimir(conjunto, 7);
}

void imprimir(float c[], int n)
{
    for(int i=0; i<n; i++)
        printf("%.2f ", c[i]);
    puts("");
}

void ordenar(float c[], int n) /*algoritmo de burbuja*/
{
    int i, j;
    //la secuencia de ordenacion debe ser repetida
    // n veces, siendo n el tamaño del array
    for(i=1; i<n; i++)
    {
        //secuencia de ordenacion
        for(j=0; j<n-1; j++)
        {
            if(c[j] > c[j+1])
                intercambio(&c[j], &c[j+1]);
        }
    }
}

void intercambio(float* x, float* y)
{
    float z;
    z = *x;
    *x = *y;
    *y = z;
}
```



```
/*Ordenacion de un array de numeros*/
Secuencia: 12.30 12.60 13.10 17.50 17.00 11.10 13.27
Secuencia ordenada: 11.10 12.30 12.60 13.10 13.27 17.00 17.50
```

Búsqueda en listas

Para la búsqueda de valores se puede utilizar la búsqueda en serie o secuencial. Su funcionamiento se basa en recorrer el algoritmo de manera secuencial hasta encontrar el elemento buscado o llegar al final de array. Mediante una función con bucle *while* condicionado y contador manual se puede conseguir el funcionamiento. La ventaja de la búsqueda en serie es que no requiere que los valores estén ordenados y la desventaja es que para listas o tablas extensas se tarda demasiado tiempo.

```
int busqueda_serie(int lista[], int TAM, int elemento)
{
    int Encontrado = 0;      //0 -> False
    int i = 0;

    while((!Encontrado) && (i <= TAM-1))
    {
        if(lista[i] == elemento)
            Encontrado = 1;
        else
            i++;
    }

    if(Encontrado)
        return i;    //retorna la posicion donde se encuentra el elemento
    else
        return -1;   //-1 significa no encontrado
}
```

Ejercicio 13

Busca la suma de los números que no pertenecen a la diagonal principal de la siguiente matriz.

4 7 -5 4 9

0 3 -2 6 -2

1 2 4 1 1

6 1 0 3 -4

Crea un programa que dibuje la matriz correctamente y que muestre dicha suma por pantalla.

```

1  #include <stdio.h>
2
3  void main()
4  {
5      int i, j, suma = 0;
6      int array[4][5] = { {4, 7, -5, 4, 9},
7                          {0, 3, -2, 6, -2},
8                          {1, 2, 4, 1, 1},
9                          {6, 1, 0, 3, -4} };
10
11     puts("\t\t/*Busca la suma de los numeros \
12 que no pertenecen a la diagonal principal*/");
13
14     printf("Matriz: \n");
15     for(i=0; i<4; i++)
16     {
17         for(j=0; j<5; j++)
18             printf("%3d ", array[i][j]);
19             //%3d -> imprime dejando 3 huecos a su izquierda
20         puts("");
21     }
22
23     puts("");
24
25     for(i=0; i<4; i++)
26         for(j=0; j<5; j++)
27             if(i != j)
28                 suma += array[i][j];
29
30     printf("La suma es %d\n", suma);
31 }
```

Ejercicio 12.- Suma de los elementos de una matriz excluyendo la diagonal principal

Ejercicio 14

Realiza un programa que sume caracteres romanos en serie, es decir, introduciendo la cadena de caracteres “XCLV” de como resultado 165 (X+C+L+V). Se debe pedir la cadena por teclado y comprobar que dicha cadena no contiene caracteres incorrectos.

Nota: Se pueden usar macros constantes (#define) para incluir los pares número romano/valor.

```
1 #include <stdio.h>
2 #define I 1
3 #define V 5
4 #define X 10
5 #define L 50
6 #define C 100
7 #define D 500
8 #define M 1000
9 void comprobacion_conversion(char r[], int d[], int*);
10 int calculo(int d[], int);
11
12 void main()
13 {
14     puts("\t\t/*Suma de caracteres romanos*/");
15
16     int resultado = 0, TAM = 0;
17     int array_decimal[51] = {0};
18     //inicializa el array de enteros a '0'
19     char array_romano[51] = {'\0'};
20     //inicializa todo el array de caracteres al valor nulo '\0'
21
22     puts("Introduce los caracteres romanos seguidos \
23 y en mayusculas (max 50 digitos)");
24     scanf("%s", &array_romano); //guarda una cadena
25
26     comprobacion_conversion(array_romano, array_decimal, &TAM);
27     resultado = calculo(array_decimal, TAM);
28
29     if(TAM > 0)
30         printf("\nEl Resultado es: %d", resultado);
31     else
32         printf("\nEl numero romano introducido es incorrecto");
33 }
```

```

34 void comprobacion_conversion(char r[], int d[], int* t)
35 {
36     for(int i=0; r[i]!='\0'; i++)
37     { //conversion
38         switch(r[i])
39         {
40             case 'I': d[i]=I; break;
41             case 'V': d[i]=V; break;
42             case 'X': d[i]=X; break;
43             case 'L': d[i]=L; break;
44             case 'C': d[i]=C; break;
45             case 'D': d[i]=D; break;
46             case 'M': d[i]=M; break;
47             default: *t = -1; //numero incorrecto
48         }
49     //comprobacion
50     if(*t == -1)
51         r[i+1] = '\0'; //cancelacion del for
52     else
53         (*t)++;
54 }
55 }

56 int calculo(int d[], int t)
57 {
58     int total = 0;
59     for(int i=0; i<t; i++)
60     {
61         total += d[i];
62     }
63     return total;
64 }
```

Ejercicio 13.- Suma de caracteres romanos

Ejercicio 15

Implementa un programa que pida por teclado un conjunto de datos y una vez introducidos calcular la media, desviación típica y varianza asociada.

Adicionalmente, agrupa el código de cálculo de la media y la varianza en funciones reutilizables.



```
1 #include <stdio.h>
2 #include <math.h>
3 float media(float c[], int t)
4 {
5     float suma=0;
6     for(int i=0; i<t; i++)
7         suma += c[i];
8     return (suma/t);
9 }
10 float varianza(float c[], int t, float med)
11 {
12     float sumatorio=0;
13     for(int i=0; i<t; i++)
14         sumatorio += pow(c[i] - med, 2);
15     return sumatorio/t;
16 }
```

```

17 void main()
18 {
19     float conjunto[51] = {0.0}, med=0.0, var=0.0;
20     int TAM=0;
21     char condicion='\0';
22     puts("\t\tCalculo de la media, desviacion y varianza \
23 de un conjunto de datos");
24
25     while(TAM<51 && condicion!='n')
26     {
27         printf("\n\xA7 %d: ", TAM+1);
28         scanf("%f", &conjunto[TAM]);
29         TAM++;
30         do
31         {
32             //Limpia el flujo de entrada que deja scanf()
33             fflush(stdin);
34             printf("Otro? SI[s] NO[n]: ");
35             scanf("%c", &condicion);
36         }while(condicion!='s' && condicion!='n');
37
38     med = media(conjunto, TAM);
39     var = varianza(conjunto, TAM, med);
40     printf("\nMedia:      %.3f", med);
41     printf("\nVarianza:   %f", var);
42     printf("\nDesviacion: %f\n", sqrt(var));
43 }

```

Ejercicio 14.- Media, varianza y desviación de un conjunto de datos

```

Calculo de la media, desviacion y varianza de un conjunto de datos
nº 1: 12.3
Otro? SI[s] NO[n]: s
nº 2: 11.2
Otro? SI[s] NO[n]: f
Otro? SI[s] NO[n]: s
nº 3: 1.5
Otro? SI[s] NO[n]: s
nº 4: 11.7
Otro? SI[s] NO[n]: n

Media:      9.175
Varianza:   19.786875
Desviacion: 4.448244

```

Salida 4.- Salida del Ejercicio 15



Ejercicio 16

Realiza un programa que tache palabras. Cada palabra que este compuesta por cuatro caracteres automáticamente será sustituida por cuatro asteriscos.

Pide una cadena de caracteres por teclado y muestra por pantalla la cadena modificada. Utiliza la función **gets()** para pedir la cadena.

```

1 #include <stdio.h>
2 int longitud(char c[]);
3 int cuenta_y_sustitucion(char c[]);
4
5 void main()
6 {
7     char condicion;
8     char cadena[50]={'\0'};
9     puts("\t\t/*Programa encargado de tratar con cadenas de caracteres*/");
10    puts("Cuenta los caracteres, cuenta las palabras de 4 letras \
11 y sustituye las palabras de 4 letras por 4 asteriscos");
12    do
13    {
14        printf("\nProporcioname un cadena de texto (max 50 caracteres): ");
15        gets(cadena);
16
17        printf("Caracteres: %d\n", longitud(cadena));
18        printf("Palabras de 4 letras: %d\n", cuenta_y_sustitucion(cadena));
19        printf("Sustitucion: %s\n\n", cadena);
20
21    do
22    {
23        printf("Otra? Si[s] No[n]: ");
24        scanf("%c", &condicion);
25        fflush(stdin); //limpia los datos residuales de entrada
26    }while(condicion!='s' && condicion!='n');
27
28 }while(condicion != 'n');
29 }
```



```

30
31 int longitud(char c[])
32 {
33     int i;
34     for(i=0; c[i]!='\0'; i++);
35     return i;
36 }
37
38 int cuenta_y_sustitucion(char c[])
39 {
40     int cont4=0, n=0;
41     for(int i=0; c[i] != '\0'; i++)
42     {
43         if(c[i] != ' ')
44         {
45             if(isalpha(c[i])) cont4++;
46             else cont4=0;
47         }
48         else cont4=0;
49
50         if(cont4 == 4 && (c[i+1] == ' ' || c[i+1] == '\0'))
51         {
52             //n = cuenta palabra de 4 letras
53             n++;
54             //sustituye palabras de 4 letras por *****
55             for(int j=0; j<4; j++) c[i+j-3]='*';
56         }
57     }
58     return n;
59 }
```

Ejercicio 15.- Tratamiento de cadenas de caracteres

```
/*Programa encargado de tratar con cadenas de caracteres*/  
Cuenta los caracteres, cuenta las palabras de 4 letras y sustituye las palabras de 4 letras por 4 asteriscos  
  
Proporcioname un cadena de texto (max 50 caracteres): En este momento, estoy desarrollando el lenguaje C  
Caracteres: 50  
Palabras de 4 letras: 1  
Sustitucion: En **** momento, estoy desarrollando el lenguaje C  
  
Otra? Si[s] No[n]: 1  
Otra? Si[s] No[n]: h  
Otra? Si[s] No[n]: s  
  
Proporcioname un cadena de texto (max 50 caracteres): mas concretamente, tratando con este programa de cadenas  
Caracteres: 56  
Palabras de 4 letras: 1  
Sustitucion: mas concretamente, tratando con **** programa de cadenas  
  
Otra? Si[s] No[n]: n
```

Salida 5.- Salida del Ejercicio 16

Ejercicio 17

Crear un programa que, mediante un array aleatorio de veinte números enteros, ordene dichos números para que la suma de los diez primeros no supere la suma de los diez últimos. Usa la función **rand()** para generar los números aleatorios.

Genera funciones reutilizables para conseguir un programa más sencillo de entender.

Nota: La función **srand(x)** cambia la serie de números enteros obtenida, pero necesita del argumento **x** que cambien con el tiempo. Resulta útil utilizar la función **time(NULL)** que devuelve (los segundos pasados desde 00:00:00 UTC, 1 de enero de 1970) un valor entero distinto que cambia con el paso del tiempo.



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 void imprimir(int v[]);
5 void generador_aleatorio(int v[]);
6 void comprobacion(int v[], int r[]);
7 void cambio_posicion_numeros(int v[]);
8
9 void main()
10 {
11     srand(time(NULL));
12
13     int vector[20]={0}, res[2]={0}, condicion=0;
14     generador_aleatorio(vector);
15
16     puts("\t\t/*Vector aleatorio de 20 numeros*/");
17     puts("La suma de los 10 primeros numeros no puede \
18 superar la suma de los 10 ultimos\n");
19     printf("Vector aleatorio: ");
20     imprimir(vector);
21     comprobacion(vector, res);
22     printf("\nSuma 10 primeros: %d\n\
23 Suma 10 ultimos: %d\n\n", res[0], res[1]);
```

```
25      do
26      {
27          if(res[0] <= res[1])
28              cambio_posicion_numeros(vector);
29          else
30              condicion = -1;
31          comprobacion(vector, res);
32      }while(condicion != -1);
33
34      printf("Vector modificado: ");
35      imprimir(vector);
36      printf("\nSuma 10 primeros: %d\n"
37 Suma 10 ultimos: %d\n", res[0], res[1]);
38 }
39
40 void imprimir(int v[])
41 {
42     for(int i=0; i<20; i++)
43     {
44         printf("%d ", v[i]);
45         if(i==9) printf("|\n");
46     }
47 }
```

```
49 void generador_aleatorio(int v[])
50 {
51     for(int i=0; i<20; i++)
52         v[i]=rand() % 51;
53     //Devuelve el resto de (rand() / 51)
54     // dando valores entre 0 y 51-1
55 }
56
57 void comprobacion(int v[], int r[])
58 {
59     r[0]=r[1]=0;
60     for(int i=0; i<20; i++)
61     {
62         if(i>=0 && i<10)    r[0]+=v[i];
63         if(i>=10 && i<20)   r[1]+=v[i];
64     }
65 }
66
```

```

67 void cambio_posicion_numeros(int v[])
68 {
69     int min_10_primeros=v[0], pos1=0, aux;
70     int max_10_ultimos=v[10], pos2=10;
71     for(int i=0; i<20; i++)
72     {
73         if(i>=0 && i<10)
74         {
75             if(v[i]<min_10_primeros)
76             {
77                 min_10_primeros = v[i];
78                 pos1 = i;
79             }
80         }
81         else if(i>=10 && i<20)
82         {
83             if(v[i]>max_10_ultimos)
84             {
85                 max_10_ultimos = v[i];
86                 pos2 = i;
87             }
88         }
89     }
90     aux = v[pos1];
91     v[pos1] = v[pos2];
92     v[pos2] = aux;
93 }
```

Ejercicio 16.- Vector aleatorio condicional

Capítulo 9. Cadenas

Introducción

En sí, el lenguaje C no tiene datos predefinidos de tipo cadena (string). En su lugar utiliza arrays de caracteres terminados en el carácter nulo '\0'. Por lo tanto, una cadena se considera un array unidimensional de tipo char. Cada carácter se guarda en un elemento del array.

Al ser una cadena un array unidimensional, esta a su vez es un puntero al primer elemento. Si utilizamos el carácter puntero (*) se obtiene el valor de un elemento. Además, se puede utilizar la aritmética para movernos por los elementos.



```

//Declaracion de cadenas
char cadena[20]; //cadena de 19 car + '\0'
//Inicializacion de cadenas
char cadena2[20] = "Esto es una cadena"; //18cars + '\0'
char cadena3[] = "Esto es otra cadena"; //19cars + '\0'
//Las comillas hacen que cree automaticamente
// el caracter nulo al final.

///Otro formato. Cadena mediante punteros
char* cad1 = "Hola"; //'H', 'o', 'l', 'a', '\0' = 5
/*cad1 es un puntero al primer caracter

//Utilizacion_1
printf("%c", *(Isidro"+4)); //Imprime (r)

char cad[] = "Hola, como estas?";
printf("\n%c == %c", *(cad+2), cad[2]); //Imprime (l == l)

//Utilizacion_2
char cad3[] = "yo estoy muy bien";

printf("%s", cad3); //el sistema envia caracteres
// a stdout(pantalla) hasta encontrar el caracter nulo

scanf("%s", cad3); //el sistema copia caracteres
// de stdin(teclado) a cad3 hasta encontrar ' ' o '\n'

gets(cad3); //el sistema copia caracteres
// de stdin(teclado) a cad3 hasta encontrar '\n'

```

Cuando se declara una cadena se debe especificar su tamaño, sin embargo, cuando se inicializa el compilador se encarga de asignar el tamaño adecuado. Si se quiere utilizar una cadena para albergar cadenas diferentes, se debe asignar un tamaño suficiente como para poder contener los diferentes tamaños.

Ahora bien, una cadena no se puede inicializar con el operador (=) después de haber sido declarada, será necesario utilizar funciones específicas del archivo de cabecera <string.h>.

Lectura y escritura de cadenas

Las funciones de lectura y escritura se encuentran en el archivo de cabecera <stdio.h>.

La lectura de cadenas se puede realizar con la función **scanf()** o con **gets()**. La diferencia principal es que scanf() captura solo la primera palabra (hasta '' o '\n') mientras que la función gets() captura varias palabras (hasta '\n'). Si existe algún error en la lectura el compilador de C devuelve NULL.

La escritura de cadenas se realiza con la función ***printf()*** o con ***puts()***. La única diferencia es que *puts()* imprime la cadena y pone el cursor en una nueva línea mientras que *printf()* solamente imprime la cadena.

Función *getchar()*

La función *getchar()* se utiliza para leer carácter a carácter. Devuelve el siguiente carácter de entrada situado en el flujo de entrada (*stdin*). En caso de error o de encontrar el fin del archivo, la función devuelve EOF que es una macro definida en *<stdio.h>*. Mediante las funciones condicionales if, while, ... y se puede comprobar fácilmente si ha habido algún error o se ha llegado al fin del archivo.

Nota: Al introducir por teclado Control + Z se envía el carácter de final de archivo al flujo de entrada (*stdin*) para una finalización manual.

Función *putchar()*

La función *putchar()* se utiliza para escribir carácter a carácter. Devuelve el siguiente carácter de salida situado en el flujo de salida (*stdout*).

```
#include <stdio.h>
#include <ctype.h>
void main()
{
    puts("\t\tPone la primera letra \
de cada palabra en mayuscula");
    char car, pre = '\n';
    while((car=getchar()) != EOF)
    {
        if(pre == ' ' || pre == '\n')
            putchar(toupper(car));
        else
            putchar(car);
        pre = car;
    }
    return 0;
}
```

<string.h>

El archivo de cabecera *<string.h>* incorpora las funciones de manipulación de cadenas utilizadas con más frecuencia. Los argumentos pasados a las funciones son recogidos como punteros al primer elemento de la cadena.

Funciones de <string.h>

memcpy()	void* memcpy(void* s1, const void* s2, size_t n); Reemplaza los primeros n bytes de s1 con los primeros n bytes de s2. Devuelve s1.
strcpy()	char* strcpy(char* dest, const char* src); Copia la cadena src a la cadena dest. Devuelve la cadena dest
strncpy()	char* strncpy(char* dest, const char* src, size_t n); Copia n caracteres de la cadena src a la cadena dest. Devuelve la cadena dest
strcat()	char* strcat (char* dest, const char* src); Añade la cadena src al final de la cadena dest. Devuelve dest
strncat()	char* strncat(char* s1, const char* s2, size_t n); Añade los primeros n caracteres de s2 a s1. Devuelve s1
strlen()	size_t strlen(const char* s); Devuelve la longitud de la cadena s
strcmp()	int strcmp(const char* s1, const char* s2); Compara la cadena s1 con s2 y devuelve '0' si s1=s2, '<0' si s1<s2 y '>0' si s1>s2
stricmp()	int stricmp(const char* s1, const char* s2); Igual a strcmp() pero sin distinguir entre mayúsculas y minúsculas
strncmp()	int strncmp(const char* s1, const char* s2, size_t n); Compara s1 con la subcadena formada por los primeros n caracteres de s2. Devuelve un entero negativo, cero o positivo si s1 es menor, igual o mayor que la subcadena de s2
strnicmp()	int strnicmp(const char* cad1, const char* cad2, size_t n); Igual a strncmp() pero sin distinguir entre mayúsculas y minúsculas
strupr()	char* strupr(char* s); Convierte la cadena s a mayúsculas
strlwr()	char* strlwr(char* s); Convierte la cadena s a minúsculas
strrev()	char* strrev(char* s); Invierte el orden de los caracteres contenidos en s. Devuelve un puntero a la cadena invertida

Ilustración 40.- Funciones de <string.h> 1

strnset()	char* strnset(char* s, int ch, size_t n); Copia n veces el carácter ch a partir de la posición inicial s[0]
atoi()	int atoi(char* cad); Convierte la cadena cad a un valor entero
atof()	double atof(const char* cad); Convierte la cadena a un valor de coma flotante
atol()	long atol(const char* cad); Convierte la cadena cad a un valor entero largo
strchr()	char* strchr(char* s1, int ch); Devuelve un puntero a la primera coincidencia del carácter sin signo ch con la cadena s1
strrchr()	char* strrchr(const char* s, int ch); Devuelve un puntero a la última coincidencia de ch en la cadena s. Devuelve NULL si ch no está en s
strspn()	size_t strspn(const char* s1, const char* s2); Devuelve el número de caracteres coincidentes de s1 con el conjunto de caracteres especificado en s2
strcspn()	size_t strcspn(const char* s1, const char* s2); Devuelve la posición en número entero de la primera coincidencia del conjunto de valores de s2 con la cadena s1
strpbrk()	char* strpbrk(const char* s1, const char* s2); Devuelve la dirección de la primera coincidencia en s1 de cualquiera de los caracteres de s2. Si no aparece ninguno devuelve NULL
strstr()	char* strstr(const char* s1, const char* s2); Busca la cadena s2 en s1 y devuelve un puntero a s1 donde se encuentra s2. Devuelve NULL en caso contrario
strtok()	char* strtok(char* s1, const char* s2); Divide la cadena s1 por el delimitador s2 en trozos (tokens). Devuelve la dirección al primer token. La llamada a strtok(NULL, s2) devuelve el puntero al siguiente token. Cuando no quedan tokens devuelve NULL

Ilustración 41.- Funciones de <string.h> 2

Asignación de cadenas

La asignación de cadenas se puede realizar de dos maneras. La primera es inicializar la variable utilizando comilla (el texto entre comillas se considera una constante de cadena de caracteres). La segunda es usar funciones específicas como strcpy() y strncpy(). La primera forma solo se puede realizar en la inicialización de la variable mientras que la segunda se puede realizar en cualquier parte del código.

Final del Temario de C

¡Enhorabuena! Has completado todos los capítulos del _Temario de C_. Ahora, es el momento de afianzar conocimientos... y que mejor manera que hacerlo mediante una serie de

problemas bien definidos. Como se suele decir - ___la mejor forma de aprender es hacer uno mismo___ -.

Te invito a echarle un vistazo a la parte de _Problemas de C_ (escribiendo “índice de los problemas”) y continuar por ahí.

Contenido de los PROBLEMAS

El contenido sobre los PROBLEMAS recoge toda la información útil que el Bot posee para afianzar el aprendizaje sobre el lenguaje C. Cada punto y problemas que se redactan a continuación están incluidos en el Bot en *tarjetas adaptativas* y han sido convertidos de formato *.docx* a *.json* de la forma que se explica en Conversión del Resumen de C a *tarjetas adaptativas*. Agradecer de nuevo a los libros de donde se ha extraído, resumido, convertido e incluso modificado esta información descritos en la Bibliografía.

Problemas. Características y organización del contenido

Características del contenido

La información proporcionada sigue en principio los siguientes elementos clave para conseguir la mayor eficacia en la resolución de problemas de C:

- **Índice: PROBLEMAS.** Muestra el índice de los PROBLEMAS dividido en partes y capítulos.
- **Problemas: para realizar.** Muestra el enunciado del problema e invita al usuario a realizarlo
- **Problemas: explicados.** Muestra el enunciado del problema y da la opción de mostrar la explicación.
- **Problemas: de solución desbloqueable.** Muestra el enunciado del problema además de una o varias cuestiones a responder. En función de los aciertos se mostrará un tipo de respuesta. Si la respuesta es correcta se mostrará la solución.

Para los problemas en los que se puede mostrar la solución, el usuario puede optar por realizarlos antes de ver la solución.

Se incluye información extra como consejos al usuario, advertencias y reglas de uso, y técnicas de programación.

Organización del contenido

Al igual que la parte del Temario y para seguir la misma línea de contenido, los problemas se han agrupado en partes y por capítulos. En este caso los problemas se sitúan **SOLO** en la segunda parte (**Fundamentos de programación en C**) aunque los temas relacionados con la primera (**Metodología de programación**) se van explicando en los problemas iniciales.

Los problemas **se han enumerado siempre de forma ascendente**, cada capítulo comenzará por un problema cuyo número será una unidad superior al último del capítulo anterior. Además, los problemas incluidos en cada capítulo serán una mezcla de los tipos citados en el apartado anterior.



PARTE II. FUNDAMENTOS DE PROGRAMACIÓN EN C

Incluye una serie de problemas para los capítulos: **Capítulo 3. Elementos básicos del lenguaje C, Capítulo 4. Operadores y expresiones, Capítulo 5. Estructuras de selección: sentencias if y switch, Capítulo 6. Estructuras repetitivas: bucles (for, while y do-while), Capítulo 7. Funciones, Capítulo 8. Arrays (listas y tablas), Capítulo 9. Cadenas.**

PARTE II. FUNDAMENTOS DE PROGRAMACIÓN EN C

Capítulo 3 y 4.

3-4.01. Problema 01. Explicar

Realice un programa que pida al usuario un entero y muestre por pantalla el valor al cuadrado y el valor al cubo.

Para conseguir una presentación clara del programa se puede seguir un orden preestablecido. Primero, declarar las variables que se van a utilizar, segundo, pedir los datos de entrada al usuario, tercero, realizar las operaciones necesarias y, por último, mostrar los datos de salida.

Entre cada apartado se dejará como mínimo una línea en blanco para su distinción. Además, se pueden incluir comentario si es preciso.

Nota: Tanto en la función *printf()* como *scanf()*, el conjunto de caracteres **%d** es similar **%i**, ambos sirven para imprimir o almacenar respectivamente un valor entero (short, int, long).

```

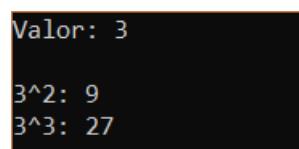
1 #include <stdio.h>
2
3 int main(void)
4 {
5     long num, cuadrado, cubo;
6
7     printf("Valor: ");
8     scanf("%d", &num);
9
10    cuadrado = num * num;
11    cubo = cuadrado * num;
12
13    printf("\n%i^2: %i\n", num, cuadrado);
14    printf("%d^3: %i\n", num, cubo);
15
16    return 0;
17 }

```

Código 4.- Problema 1

<https://i.ibb.co/xfw9sgV/3-4-01.png>

<https://github.com/isidropal/EjerciciosC/blob/master/3-4.01.c>



```

Valor: 3
3^2: 9
3^3: 27

```

Salida 6.- Problema 1

<https://i.ibb.co/Jcy2MGt/3-4-01E.png>

3-4.02. Problema 02. Desbloquear.

Escribir un programa que pida al usuario los valores de la base y de la altura de un triángulo y que muestre por pantalla el valor de la superficie.

¿Qué tipo de dato será el más adecuado para representar la base, la altura y la superficie del triángulo?

Real V, Entero

¡Efectivamente! El valor más adecuado para representar una longitud y una superficie siempre será un tipo real (float, double); aunque el tipo de dato *float* sería suficiente, en arquitecturas modernas de ordenadores es mejor utilizar siempre *double* para retrasar la aparición de problemas de redondeo.

No sería lo más adecuado. Aunque se podría pensar que la base y la altura se podrían recoger en un tipo entero (short, int, long), la propia formula de la superficie nos arrojaría valores reales en la mayoría de los casos.

```

1 #include <stdio.h>
2 void main()
3 {
4     double base, alt, sup;
5
6     printf("Base: ");
7     scanf("%lf", &base);
8     printf("Altura: ");
9     scanf("%lf", &alt);
10
11    sup = base * alt / 2;
12
13    printf("La superficie del triangulo");
14    //se representan dos decimales -> "% .2f"
15    printf(" (B: %.2f, A: %.2f) es %.2f", base, alt, sup);
16 }
```

Código 5.- Problema 2

<https://i.ibb.co/hc2qqbg/3-4-02.png>

<https://github.com/isidrop/C/blob/master/3-4.02.c>

```
Base: 1.25
Altura: 2.36
La superficie del triangulo (B: 1.25, A: 2.36) es 1.47
```

Salida 7.- Problema 2

<https://i.ibb.co/Nr426Zg/3-4-02E.png>

Aviso: Los ordenadores actuales tienen una gran potencia, por lo tanto, lo más conveniente es utilizar los tipos de dato más extensos; long para números entero, double para número real y char para carácter.



3-4.03. Problema 03. Explicar.

Escriba un programa que solicite el radio de una circunferencia y muestre por pantalla la longitud de la circunferencia y la superficie que abarca.

En este ejercicio vamos a realizar una mezcla de tipos de dato para afianzar conocimientos. Se recogerá el radio como tipo de dato entero. Para la longitud y la superficie se cogerá un tipo real.

Entonces, la inclusión del número Pi en los cálculos de la longitud y la superficie, por ser multiplicaciones, forzarán al radio a convertirse implícitamente a tipo real.

El número Pi se puede definir como una constante define (`#define`) o como una constante float (`const float`).

```

1  #include <stdio.h>
2  #define PI 3.14159
3  void main(void)
4  {
5      short R;
6      float L, S;
7      const float pi = 3.14159;
8
9      printf("Radio: ");
10     // "%hd" = short, aunque tambien vale "%d"
11     scanf("%hd", &R);
12
13     L = 2 * PI * R;
14     S = pi * R * R;
15
16     printf("Longitud: %.3f", L);
17     printf("\nSuperficie: %.3f", S);
18 }
```

Código 6.- Problema 3

<https://i.ibb.co/Vp555Tg/3-4-03.png>

<https://github.com/isidropo/EjerciciosC/blob/master/3-4.03.c>



```
Radio: 9
Longitud: 56.549
Superficie: 254.469
```

Salida 8.- Problema 3

<https://i.ibb.co/zmStsb4/3-4-03E.png>

3-4.04. Problema 04. Realizar.

Realice un programa que sirva de conversor de temperaturas desde Fahrenheit hacia Celsius.
Para ello utilice la ecuación $Celsius = (5/9) * (Fahrenheit - 32)$.

Aviso: La ecuación de conversión de temperatura escrita tal cual, realiza un error de truncamiento produciendo un '0' como resultado. El error está localizado en (5/9), C entiende el cociente de enteros (5/9 = 0.55) como un entero y lo trunca a '0'. Por lo tanto, se debe utilizar 5.0, 9.0, (double) 5 o (double) 9; con cualquiera de ellos es suficiente.

3-4.05. Problema 05. Desbloquear.

¿Qué valor tomarán las variables del siguiente cuadro?

```
1 int a = 10, b = 4, c, d;
2 double x = 10.0, y = 4.0, z, t, v;
3
4 c = a / b;
5 d = x / y;
6 z = a / b;
7 z = x / y;
8 t = (1/2) * x;
9 v = (1.0/2) * x;
```

Código 7.- Problema 5

<https://i.ibb.co/hKq2kKW/3-4-05.png>

c = 2.5, d = 2, z = 2, z = 2.5, t = 0, v = 5

c = 2, d = 2, z = 2.0, z = 2.5, t = 0.0, v = 5.0 V

c = 2, d = 2.0, z = 2.0, z = 2, t = 0.0, v = 5

¡Correcto!

¡Incorrecto! Le invito a que escriba el programa en su compilador y compruebe la solución.

3-4.06. Problema 06. Explicar.

¿Qué mostrará por pantalla el siguiente programa?



```

1 #include <stdio.h>
2 #include <stdint.h>
3 void main()
4 {
5     int8_t a = 127;
6     a++;
7     printf ("%hd", a);
8 }
```

Código 8.- Problema 6

<https://i.ibb.co/vqwgsDM/3-4-06.png>

Pues bien... en primer lugar, hay que decir que el tipo *int8_t* (incluido en *<stdint.h>*) tiene un rango desde -128 a 127. Por lo tanto, si se aumentase en una unidad el número 127, este desbordaría pasando al límite inferior y dando como resultado **-128**.

Esto ocurre con cada tipo de dato, es decir, si se excediese alguno de los límites de cualquier tipo de dato este desbordaría hacia su límite contrario.

3-4.07. Problema 07. Realizar.

Realiza un programa que muestre por pantalla el tamaño en bits y bytes de los tipos de datos *short*, *int* y *long*.

Nota: El operador *sizeof(x)* devuelve el tamaño de *x* en bytes como un entero.

3-4.08. Problema 08. Explicar.

Al elevar al cuadrado el número Áureo se obtiene el mismo valor que al sumarle uno, es decir, se cumple la siguiente ecuación $x * x = x + 1$. Mediante esta ecuación se va a realizar un programa que calcule el número Áureo.

Si se despeja *x* se obtiene $x = (1 + \sqrt{5}) / 2$. El cálculo es bastante sencillo, lo único que nos faltaría seria la función de cálculo de la raíz cuadrada que en C se realiza con *sqrt(x)* y está incluida en *<math.h>*.

```
1 #include <stdio.h>
2 #include <math.h>
3 int main(void)
4 {
5     double aureo;
6
7     printf("Obtener el numero Aureo mediante \
8 la ecuacion que lo describe (aureo^2 = aureo + 1)");
9
10    aureo = (1 + sqrt(5)) / 2;
11
12    printf("\nNumero Aureo = %.15f", aureo);
13    return 0;
14 }
```

Código 9.- Problema 8

<https://i.ibb.co/rcVvQqK/3-4-08.png>

<https://github.com/isidrop/C/blob/master/3-4.08.c>

Obtener el numero Aureo mediante la ecuacion que lo describe (aureo² = aureo + 1)
Número Aureo = 1.618033988749895

Salida 9.- Problema 8

<https://i.ibb.co/mF0YqQS/3-4-08E.png>

3-4.09. Problema 09. Realizar.

Crear un programa que pida un número no negativo menor que 100.000 y que lo muestre por pantalla separado, es decir, para el numero 12430 debe mostrar **1 2 4 3 0**. Este problema se resuelve mediante cálculos matemáticos.

3-4.10. Problema 10. Desbloquear.

Según el código que se muestra abajo.



```

1 #include <stdio.h>
2 #include <stdint.h>
3 void main()
4 {
5     int16_t a = 0xFFFF4, b = 12, c = a == -b ? 1 : 0;
6     printf ("%hd\t%hd\t%hd", a, b, c);
7 }
```

Código 10.- Problema 10

<https://i.ibb.co/8XrFM2P/3-4-10.png>

¿Qué valor se mostrará al imprimir *a*?

-12 V, 31, 23

¡Eso es! Al realizar, compilar y ejecutar el código se comprueba que *a* vale -12. Si el tipo de la variable fuese *int* (equivalente a *int32_t*) el resultado sería el mismo, simplemente se muestra el tipo *int16_t* para hacer entender que existen otros tipos de datos.

¡No! Si realiza, compila y ejecuta el código se comprueba que *a* vale -12. Si el tipo de la variable fuese *int* (equivalente a *int32_t*) el resultado sería el mismo, simplemente se muestra el tipo *int16_t* para hacer entender que existen otros tipos de datos.

3-4.11. Problema 11. Desbloquear.

¿Qué salida mostrará el código de abajo?

```

1 #include <stdio.h>
2 #include <stdint.h>
3 int main()
4 {
5     int16_t a = 0xFFFF;
6     uint16_t b = 0xFFFF;
7
8     printf ("%s", a == b ? "IGUALES" : "DISTINTOS");
9
10    return 0;
11 }
```

Código 11.- Problema 11

<https://i.ibb.co/qWvzJGJ/3-4-11.png>

IGUALES V, DISTINTOS

¡Cierto! Como el tipo sin signo no abarca los números negativos y 0xFFFF se sitúa en el desbordamiento, estos no serán iguales ($a = -1$, $b = 65535$).

¡No! Si usted comprueba la ejecución del programa verá que el tipo sin signo no abarca los números negativos y 0xFFFF se sitúa en el desbordamiento, por lo tanto, no serán iguales.

3-4.12. Problema 12. Explicar.

Se requiere la construcción de un programa que muestre por pantalla el código ASCII de un carácter introducido por teclado.

En primer lugar, guardaremos el carácter en el código de formato %c y finalmente para conseguir el resultado deseado se deberá incluir en la cadena constante de *printf()* el código de formato %hd que muestra el entero corto (1 byte) del carácter.

```

1 #include <stdio.h>
2 #include <stdint.h>
3 int main()
4 {
5     int8_t car;
6
7     printf("Introduzca un caracter para mostrar su \
8 codigo ASCII: ");
9     scanf("%c", &car);
10
11    printf("El codigo ASCII del caracter %c en decimal \
12 es %hd", car, car);
13    return 0;
14 }
```

Código 12.- Problema 12

<https://i.ibb.co/0BrnShZ/3-4-12.png>

<https://github.com/isidropal/EjerciciosC/blob/master/3-4.12.c>

```
Introduzca un caracter para mostrar su codigo ASCII: u
El codigo ASCII del caracter u en decimal es 117
```

Salida 10.- Problema 12

<https://i.ibb.co/Xs4Fm5P/3-4-12E.png>

3-4.13. Problema 13. Realizar

Escribe un programa como el mostrado más abajo y comprueba cual es la salida por pantalla.

```

1 #include <stdio.h>
2 #define _USE_MATH_DEFINES
3 #include <math.h>
4 void main()
5 {
6     double x = M_PI;
7     printf("Pi con 1 decimal %21.1f\n", x);
8     printf("Pi con 2 decimal %21.2f\n", x);
9     printf("Pi con 3 decimal %21.3f\n", x);
10    printf("Pi con 4 decimal %21.4f\n", x);
11    printf("Pi con 5 decimal %21.5f\n", x);
12    printf("Pi con 6 decimal %21.6f\n", x);
13    printf("Pi con 7 decimal %21.7f\n", x);
14    printf("Pi con 8 decimal %21.8f\n", x);
15    printf("Pi con 9 decimal %21.9f\n", x);
16    printf("Pi con 10 decimal %20.10f\n", x);
17    printf("Pi con 11 decimal %20.11f\n", x);
18    printf("Pi con 12 decimal %20.12f\n", x);
19    printf("Pi con 13 decimal %20.13f\n", x);
20    printf("Pi con 14 decimal %20.14f\n", x);
21    printf("Pi con 15 decimal %20.15f\n", x);
22 }

```

Código 13.- Problema 13

<https://i.ibb.co/3sSxKsp/3-4-13.png>

Aviso: Según el compilador se deberá incluir `#define _USE_MATH_DEFINES` previamente al archivo de cabecera `<math.h>` para poder utilizar las constantes de dicho archivo, en este caso `M_PI`.

Capítulo 5.

5.01. Problema 14. Realizar.

Solicite al usuario que introduzca dos valores del mismo tipo en dos variables distintas y a continuación intercambíalos y muéstralos por pantalla.



5.02. Problema 15. Realizar.

Se quiere obtener la superficie de un triángulo rectángulo, para ello se debe introducir por teclado la *base* y la *altura*. Primero... se calculará la superficie para una base y altura de tipo entero (*long*). Segundo... se calculará la superficie para una base y altura de tipo real (*double*). En ambos casos la superficie debe ser tipo real (*double*).

5.03. Problema 16. Explicar.

Se precisa el cálculo de la longitud, el área y el volumen de una circunferencia, por lo tanto, solamente es necesaria la obtención de su radio.

Para comenzar escribiremos qué hará nuestro programa, a continuación, solicitaremos el dato necesario (radio) y por último haremos los cálculos y mostraremos los resultados.

```

1 #include <stdio.h>
2 #define PI 3.14159265
3 void main()
4 {
5     double r, resul;
6
7     printf("\tCalculo de la longitud, area y volumen");
8     printf(" de una circunferencia de radio dado\n");
9     printf("Radio: ");
10    scanf("%lf", &r);
11
12    resul = 2 * PI * r;
13    printf("\nLong:\t%f\n", resul);
14    resul = PI * r * r;
15    printf("Area:\t%f\n", resul);
16    resul = 4 * PI * r * r * r / 3;
17    printf("Vol:\t%f", resul);
18 }
```

Código 14.- Problema 16

<https://i.ibb.co/rGnd3dW/5-03.png>

<https://github.com/isidropo/EjerciciosC/blob/master/5.03.c>

```

        Calculo de la longitud, area y volumen de una circunferencia de radio dado
Radio: 6.27

Long: 39.395572
Area: 123.505118
Vol: 1032.502784
```

Salida 11.- Problema 16

<https://i.ibb.co/S6fZVqg/5-03E.png>

En este caso hemos querido reducir la cantidad de variables utilizadas en el programa, utilizando solo una para mostrar todos los resultados.

Aviso: En el momento en que un programa empiece a extenderse la cantidad de variables utilizadas jugará un papel importante en su comprensión y extensión, dicho esto, es aconsejable utilizar las variables necesarias... **ni más ni menos**.

5.04. Problema 17. Desbloquear.

Se requiere realizar un programa que sepa de cuántos dígitos se compone el número que el usuario ha introducido previamente.

¿Es posible realizar este programa con los conocimientos adquiridos hasta ahora, es decir, sin utilizar estructuras en bucle?

Sí, No V

A priori parece que si se puede conseguir, pero una vez comienzas, se ve claramente que solo se puede hacer para un número determinado de dígitos. Según lo que se ha visto hasta ahora, se podría realizar mediante *secuenciación*, es decir, una instrucción detrás de otra. Eso sí, habría que limitar el programa a un número fijo de dígitos.

¡Por supuesto! No utilizar estructuras en bucle limitaría en gran medida el programa, ya que utilizando instrucciones en *secuencia* (una instrucción detrás de otra) solo se conseguiría hacerlo para un número determinado de dígitos.

```
1 #include <stdio.h>
2 void main()
3 {
4     long num;
5     printf("Introduce un numero de tres digitos: ");
6     scanf("%d", &num);
7
8
9     printf("Digitos: %d", num / 100);
10    num = num % 100;
11    printf(" %d", num / 10);
12    num = num % 10;
13    printf(" %d", num);
14 }
```

Código 15.- Problema 17

<https://i.ibb.co/KKkjmjc/5-04.png>

<https://github.com/isidropo/EjerciciosC/blob/master/5.04.c>

```
Introduce un numero de tres digitos: 565
Digitos: 5 6 5
```

Salida 12.- Problema 17

<https://i.ibb.co/cw1Vw1y/5-04E.png>

5.05. Problema 18. Explicar.

Restricción: NO USE **else** solo **if** o **if-else if**.

Crear un programa que orden los valores de tres variables enteras. En primer lugar, el programa debe solicitar al usuario a introducción de tres valores en tres variables enteras (x_1 , x_2 , x_3). En segundo lugar, se deben ordenar los valores de tal forma que la variable x_1 almacene el valor más pequeño, mientras que x_3 tenga el mayor. La última condición es que ninguno de los valores sea igual.

Hay que decir que este problema es más bien matemático, y que en el mundo de C no es muy utilizado. El problema se agrava más aun con la limitación de herramientas que tenemos hasta ahora (operadores lógicos, funciones, ...).

Dicho esto, se muestra una posible solución que compara las variables de izquierda a derecha y en algunos casos derecha a izquierda. Si se cumple la comparación se intercambian los valores comparados.



```

1 #include <stdio.h>
2 void main()
3 {
4     long x1, x2, x3, aux;
5     printf ("\tIntroduce tres valores a ordenar\n");
6
7     printf ("x1 = "); scanf ("%d", &x1);
8     printf ("x2 = "); scanf ("%d", &x2);
9     printf ("x3 = "); scanf ("%d", &x3);
10
11    if(x1 > x2)
12    {
13        aux = x1; x1 = x2; x2 = aux;
14        if(x2 > x3)
15        {
16            aux= x2; x2 = x3; x3 = aux;
17            if(x1 > x2)
18            {
19                aux = x1; x1 = x2; x2 = aux;
20            }
21        }
22    }
23    else if(x1 > x3)
24    {
25        aux = x1; x1 = x3; x3 = aux;
26        if(x2 > x3)
27        {
28            aux = x2; x2 = x3; x3 = aux;
29        }
30    }
31    else if(x2 > x3)
32    {
33        aux = x2; x2 = x3; x3 = aux;
34    }
35
36    printf ("\nx1 = %d, x2 = %d, x3 = %d", x1, x2, x3);
37 }

```

Código 16.- Problema 18

<https://i.ibb.co/7rCRWCJ/5-05.png>

<https://github.com/isidropo/EjerciciosC/blob/master/5.05.c>

```
Introduce tres valores a ordenar
x1 = 21
x2 = 23
x3 = 16
```

Salida 13.- Problema 18

<https://i.ibb.co/T2d0sLj/5-05E.png>

5.06. Problema 19. Explicar.

Restricción: NO USE **else** solo **if** o **if-else if**.

Si se repitiese el ejercicio anterior, pero en lugar de ordenar 3 variables hubiese que ordenar 4, ¿cómo se haría?

Pues efectivamente se haría de la misma forma, comparando e intercambiando hasta conseguir la ordenación. Esto supondría un gran aumento de la cantidad de comparaciones a realizar volviendo el problema muy engorro y poco entendible.

Como se dijo en el problema anterior esta clase de problemas de ordenación son más bien matemáticos.

5.07. Problema 20. Realizar.

Restricción: NO USE **else** solo **if** o **if-else if**.

Crear un programa que asigne a una variable el menor de los valores de otras **3** variables introducidas por teclado y la muestre por pantalla.

5.08. Problema 21. Realizar.

Restricción: NO USE **else** solo **if** o **if-else if**.

Crear un programa que asigne a una variable el menor de los valores de otras **4** variables introducidas por teclado y la muestre por pantalla.

5.09. Problema 22. Realizar.

Restricción: Use **if-else if....-else** pero NO USE los operadores lógicos AND (**&&**), OR (**//**) ni NOT (**!**).

Realizar un programa que muestre por pantalla **3** variables ordenadas según su valor introducido por teclado. Por ejemplo, para las variables $x1=2$, $x2=1$, $x3=5$ se debe mostrar $x2 < x1 < x3$.

5.10. Problema 23. Explicar.

Restricción: Use **if-else if....-else** pero NO USE los operadores lógicos AND (**&&**), OR (**//**) ni NOT (**!**).

Realizar un programa que muestre por pantalla **4** variables ordenadas según su valor introducido por teclado. Por ejemplo, para las variables $x1=2$, $x2=1$, $x3=5$, $x4=4$ se debe mostrar $x2 < x1 < x4 < x3$.



Si ha conseguido realizar el programa anterior se habrá dado cuenta que realizar este es aún más complejo, por ende, no es necesario su realización.

Ejercicio 5.09:

```

1 #include <stdio.h>
2 void main()
3 {
4     long x1, x2, x3;
5     printf("x1 = "); scanf("%d", &x1);
6     printf("x2 = "); scanf("%d", &x2);
7     printf("x3 = "); scanf("%d", &x3);
8
9     if(x1 > x2)
10    {
11        if(x2 > x3)
12            printf("x3 < x2 < x1"); //1!
13        else if(x1 > x3)
14            printf("x2 < x3 < x1"); //3!
15        else
16            printf("x2 < x1 < x3"); //2
17    }
18    else if(x1 > x3)
19        printf("x3 < x1 < x2"); //2!
20    else if(x2 > x3)
21        printf("x1 < x3 < x2"); //3
22    else
23        printf("x1 < x2 < x3"); //1
24 }
```

Código 17.- Problema 23

<https://i.ibb.co/1nCs7rn/5-09.png>

```
x1 = 2
x2 = 6
x3 = 3
x1 < x3 < x2
```

Salida 14.- Problema 23

<https://i.ibb.co/hLXJyj8/5-09E.png>

Como conclusión se puede decir que sin el uso de otras herramientas de C como los operadores lógicos un programa de este tipo se puede volver una odisea.

5.11. Problema 24. Desbloquear.

Restricción: Use *if-else if-...-else* pero NO USE los operadores lógicos AND (**&&**), OR (**//**) ni NOT (**!**).

Se requiere resolver la ecuación de primer grado, $ax + b = 0$, mediante la introducción por teclado de los datos a y b .

Las reglas matemáticas prohíben un valor para $a = 0$ (en caso de querer obtener el valor buscado x). Entonces, siendo esta la restricción sumada a no poder usar operadores lógicos. ¿Se podría resolver el problema mediante la estructura de selección *switch*?

Si V, No

¡Por supuesto! En realidad, solo tendríamos que recurrir una vez más a las matemáticas. Un numero distinto de 0 es lo mismo que decir que un número siempre será mayor y menor que 0. En este caso utilizaremos el selector *switch* cuya **variable de control** nos proporcionará dos casos posibles (0, 1) mediante el operador condicional (**?:**).

¡No! Llegar a la conclusión puede resultar algo complicado, pero se puede conseguir. Se debe crear una estructura *switch* cuya **variable de control** varíe en función de la condición impuesta, o lo que es lo mismo, hacer uso del operador condicional (**?:**).

```

1 #include <stdio.h>
2 void main()
3 {
4     double a, b;
5     printf("\tProporcionar los coeficientes a y b de la ");
6     printf("ecuacion de 1\xA7 grado, ax + b = 0\n");
7     printf("a: "); scanf("%lf", &a);
8     printf("b: "); scanf("%lf", &b);
9
10    switch((a>0 || a<0) ? 0 : 1)
11    {
12        case 0:
13            printf("El resultado es x = %.2f", -b/a);
14            break;
15        default:
16            printf("ERROR. 'a' debe ser distinta de 0");
17    }
18 }
```

Código 18.- Problema 24

<https://i.ibb.co/4MZ1h6X/5-11.png>

<https://github.com/isidropo/EjerciciosC/blob/master/5.11.c>

```

Proporcionar los coeficientes a y b de la ecuacion de 1º grado, ax + b = 0
a: 2.23
b: 1.98
```

Salida 15.- Problema 24

<https://i.ibb.co/nrKhHW7/5-11E.png>

5.12. Problema 25. Explicar.

Realice un programa que muestre por pantalla **3** variables enteras ordenadas según su valor introducido por teclado.

Se trata de hacer el mismo ejercicio realizado anteriormente, pero usando todas las herramientas disponibles en este tema (estructura *if-else if-...-else* y operadores lógicos). Ahora si, en cada sentencia de selección pondremos las condiciones necesarias junto a los operadores lógicos y conseguiremos fácilmente todos los resultados.

```

1 #include <stdio.h>
2 void main()
3 {
4     long x1, x2, x3;
5     printf ("x1 = "); scanf ("%d", &x1);
6     printf ("x2 = "); scanf ("%d", &x2);
7     printf ("x3 = "); scanf ("%d", &x3);
8
9     if(x1 > x2 && x2 > x3)
10        printf ("x3 < x2 < x1");
11    else if(x1 > x2 && x1 > x3)
12        printf ("x2 < x3 < x1");
13    else if(x1 > x2 && x2 < x3 && x1 < x3)
14        printf ("x2 < x1 < x3");
15    else if(x1 > x3)
16        printf ("x3 < x1 < x2");
17    else if(x2 > x3)
18        printf ("x1 < x3 < x2");
19    else
20        printf ("x1 < x2 < x3");
21 }

```

Código 19.- Problema 25

<https://i.ibb.co/18YHrsC/5-12.png>

<https://github.com/isidrop/C/blob/master/5.12.c>

```

x1 = 232
x2 = 896
x3 = 467
x1 < x3 < x2

```

Salida 16.- Problema 25

<https://i.ibb.co/2WJ6Kyx/5-12E.png>

5.13. Problema 26. Realizar.

Repite el ejercicio anterior, pero en lugar de representar ordenadamente tres variables introducidas por teclado, que sean **4** variables.

5.14. Problema 27. Explicar.

Crear un programa que indique ***si un punto sobre un plano (x, y) está dentro, sobre o fuera*** de una circunferencia con centro y radio dados. El centro y el radio deberá ser introducido por teclado, además del punto que se quiere localizar. Todos los datos deberán ser enteros.

Para solucionar el problema de la forma más sencilla debemos evitar mover el centro de la circunferencia del centro de coordenadas (0, 0); en su lugar moveremos el valor del punto a localizar. De esta manera conseguiremos que con una estructura ***if-else if-else*** se sepa si el punto a localizar está dentro, en el borde o fuera de la circunferencia.

```

1 #include <stdio.h>
2 void main()
3 {
4     long px, py, cx, cy, r;
5     printf("Punto:\nPx: "); scanf("%hd", &px);
6     printf("Py: "); scanf("%hd", &py);
7     printf("\nCentro:\nCx: "); scanf("%hd", &cx);
8     printf("Cy: "); scanf("%hd", &cy);
9     printf("\nRadio:\t"); scanf("%hd", &r);
10
11     long pmod_x = px - cx, pmod_y = py - cy;
12
13     if( (pmod_x == r && pmod_y == 0) || |
14         (pmod_x == 0 && pmod_y == r) || |
15         (pmod_x == -r && pmod_y == 0) || |
16         (pmod_x == 0 && pmod_y == -r) )
17         printf("\nPunto en el BORDE de la circunferencia.");
18     else if( (pmod_x > -r && pmod_x < r) &&
19             (pmod_y > -r && pmod_y < r) )
20         printf("\nPunto DENTRO de la circunferencia.");
21     else
22         printf("\nPunto FUERA de la circunferencia.");
23 }
```

Código 20.- Problema 27

<https://i.ibb.co/zmWdXcK/5-14.png>

<https://github.com/isidropo/EjerciciosC/blob/master/5.14.c>

```
Punto:  
P_x: 5  
P_y: 4  
  
Centro:  
c_x: 2  
c_y: 3  
  
Radio: 3  
  
Punto DENTRO de la circunferencia.
```

Salida 17.- Problema 27

<https://i.ibb.co/Hp0c0cJ/5-14E.png>

5.15. Problema 28. Desbloquear.

Realizar un programa que muestre por pantalla la ecuación de la recta $y = m * x + n$ mediante la introducción por pantalla las coordenadas de dos puntos $p1 = (x_1, y_1)$ y $p2 = (x_2, y_2)$. Obtén los valores de m y n y por último muestra la ecuación de la recta.

Dos requisitos, los dos puntos no pueden ser el mismo y el valor de x de ambos tampoco.

¿Cuántas variables como mínimo se necesitarán para recoger los datos de entrada?

3, 5, 2, 4 V

Efectivamente, **4** serán las variables necesarias para recoger los datos de entrada, dos para cada punto. Este problema es simplemente matemático, mediante una ecuación para cada punto formaremos un sistema de dos ecuaciones del cual despejaremos las variables m y n .

No hay forma posible de realizar un cálculo matemático que necesita de 4 variables, dos para cada punto, usando una cantidad inferior. Del mismo modo tampoco se necesitan más de 4.

```

1 #include <stdio.h>
2 void main()
3 {
4     double px1, py1, px2, py2, m, n;
5     printf("Puntos:\nP1_x: "); scanf("%lf", &px1);
6     printf("P1_y: "); scanf("%lf", &py1);
7     printf("\nP2_x: "); scanf("%lf", &px2);
8     printf("P2_y: "); scanf("%lf", &py2);
9
10    m = (py1-py2) / (px1-px2);
11    n = py1 - m * px1;
12
13    printf("\ny = %.2f x + %.2f", m, n);
14 }
```

Código 21.- Problema 28

<https://i.ibb.co/jyvbjdQ/5-15.png>

<https://github.com/isidrop/C/blob/master/5.15.c>

```

Puntos:
P1_x: 7.56
P1_y: 4.32

P2_x: 3.3
P2_y: 9.3

y = -1.17 x + 13.16
```

Salida 18.- Problema 28

<https://i.ibb.co/L8WqNf5/5-15E.png>

5.16. Problema 29. Explicar.

Desarrolla un programa que realice la ecuación de 2º grado, $ax^2 + bx + c = 0$, pidiendo al usuario los valores a , b y c por teclado. Entre las diferentes soluciones se requiere que se expresen tanto las soluciones reales como las posibles soluciones complejas. Si los valores de a y b introducidos son 0 el programa debe mostrar que ambos valores no pueden ser cero a la vez.

Bien... según el enunciado mostrado las condiciones son: soluciones reales o complejas y ambas a y b no pueden ser cero. Haciendo un estudio previo se puede ver que a o b pueden ser cero; obteniendo respectivamente dos o una solución.



Entonces, en primer lugar, se comprobará si alguna a o b es cero, es decir, si ambas son distintas de 0, a es distinta de 0 o b es distinta de 0. Para cada caso y, en segundo lugar, se comprobará si las soluciones serán reales o complejas, es decir, si el número dentro de la raíz cuadrada (ecuación de 2º grado) es negativo.

Por lo tanto, una estructura *if-else if-...* sumada a estructuras internas *if-else* es más que suficiente. Está claro que los tipos de dato serán reales.

```

1 #include <stdio.h>
2 #include <math.h>
3 void main()
4 {
5     double a, b, c, delta;
6     puts("\tCalculadora ecuaciones de 2 grado,\ \
7 dame los valores de los coeficientes a, b y c");
8     printf("a b c: "); scanf("%lf %lf %lf", &a, &b, &c);
9
10    if(a!=0 && b!=0 && c!=0)
11    {
12        delta = b*b - 4*a*c;
13        if(delta >= 0)
14            printf("x1 = %.4f\nx2 = %.4f",
15                   (-b + sqrt(delta)) / (2*a),
16                   (-b - sqrt(delta)) / (2*a));
17        else
18            printf("x1 = %.4f + %.4f i\nx2 = %.4f - %.4f i",
19                   -b/(2*a), sqrt(-delta/(4*a*a)),
20                   -b/(2*a), sqrt(-delta/(4*a*a)));
21    }
22    else if(a!=0 && b==0)
23    {
24        delta = -c/a;
25        if(delta >= 0)
26            printf("x1 = %.4f\nx2 = %.4f", sqrt(delta),
27                   -sqrt(delta));
28        else
29            printf("x1 = %.4f i\nx2 = %.4f i", sqrt(-delta),
30                   -sqrt(-delta));
31    }
32    else if(a==0 && b!=0)
33        printf("x = %.3f", -c/b);
34    else if(a==0 && b==0)
35        printf("a y b no pueden ser 0");
36 }

```

Código 22.- Problema 29

<https://i.ibb.co/M1w8Dmg/5-16.png>

<https://github.com/isidropo/EjerciciosC/blob/master/5.16.c>

```
Calculadora ecuaciones de 2 grado, dame los valores de los coeficientes a, b y c
a b c: 1.1 2.8 1.45
x1 = -0.7235
x2 = -1.8220
```

Salida 19.- Problema 29

<https://i.ibb.co/d4hGj2X/5-16E.png>

Capítulo 6.

6.01. Problema 30. Desbloquear.

Realizar un programa que muestre por pantalla la tabla de multiplicar del número introducido por teclado. Utilice el bucle *for*.

¿Cuántas variables se necesitan como mínimo para cumplir con el cometido?

1, 2 V, 3

¡Eso es! Con solo 2 variables es más que suficiente. Una de las variables se utiliza para recoger el valor introducido por teclado y la otra para recorrer el bucle *for*. En cada iteración del bucle *for* conseguimos una multiplicación hasta conseguir las 10 necesarias.

¡Error! Con 1 sola variable es imposible... si o si se necesita una variable como almacén para el dato introducido por teclado y otra que sirva de iteración para el bucle *for*.

¡Error! Puede parecer que usar 3 variables es lo correcto, pero hay que decir que con solo 2 es suficiente; una variable como almacén para el dato introducido por teclado y otra que sirva de iteración para el bucle *for*.

```
1 #include <stdio.h>
2 void main()
3 {
4     long num;
5     printf("Tabla de multiplicar del numero: ");
6     scanf("%d", &num);
7
8     for(int i=0; i<11 ; i++)
9         printf("%d x %2d = %2d\n", num, i, num*i);
10 }
```

Código 23.- Problema 30

<https://i.ibb.co/x1v0D2D/6-01.png>

<https://github.com/isidropo/EjerciciosC/blob/master/6.01.c>

```
Tabla de multiplicar del numero: 23
23 x  0 =  0
23 x  1 = 23
23 x  2 = 46
23 x  3 = 69
23 x  4 = 92
23 x  5 = 115
23 x  6 = 138
23 x  7 = 161
23 x  8 = 184
23 x  9 = 207
23 x 10 = 230
```

Salida 20.- Problema 30

<https://i.ibb.co/LxJ0M3W/6-01E.png>

6.02. Problema 31. Realizar.

Realiza un programa que muestre por pantalla los impares menores de 200 alineados en tres columnas. Haz uso de los códigos de escape: tabulador (\t) y nueva línea (\n) en la función *printf()* para conseguir alinear los valores en tres columnas.

6.03. Problema 32. Realizar.

Crear un programa que calcule el término n-ésimo de la serie de Fibonacci para ello pida al usuario la introducción de dicho término.

Nota: Los primero dos números de la sucesión de Fibonacci son sucesivamente 0 y 1. Los números posteriores serán la suma del actual más el anterior. Por ejemplo: $f_0=0, f_1=1, f_2=1+0=1, f_3=1+1=2, \dots$

6.04. Problema 33. Desbloquear.

Se requiere crear un programa que calcule la división de dos enteros mediante el método de *restas sucesivas*. Dicho método se encarga de obtener el **cociente** mediante la resta del dividendo y el divisor, es decir, se restará el dividendo menos el divisor y el resultado se volverá a restar con el divisor recurrentemente hasta que el resultado sea menor que el divisor; entonces el número de veces que se haya realizado la resta será igual al **cociente**.

¿Cuántas restas sucesivas hay que realizar para obtener el resultado de la división 1234 / 23?

52, 53 V, 58, 55

¡Correcto! Ese sería el resultado. Por seguridad se deben inicializar todas las variables en su declaración; al valor de otra variable, mediante *scanf()* o a un valor nulo (0 para números y \0 para caracteres y cadenas). Por ejemplo, si en este caso no inicializásemos a 0 la variable *coef* conseguiríamos resultados inesperados.

¡Ese no es el resultado! En el caso de haber realizado este sencillo ejercicio correctamente o de haberlo realizado... conseguirías otro resultado diferente. Te invito a que lo hagas/corrijas, es



tan fácil como realizar correctamente un bucle *for* (funcionamiento de dicha estructura en la teoría).

```

1 #include <stdio.h>
2 void main()
3 {
4     long a, b, coef=0;
5     printf ("\tMetodo de division mediante \
6 restas sucesivas\n");
7     printf ("a: "); scanf ("%d", &a);
8     printf ("b: "); scanf ("%d", &b);
9
10    for(a; a >= b; a=a-b)
11        coef++;
12
13    printf ("\nCoeficiente: %d", coef);
14 }
```

Código 24.- Problema 33

<https://i.ibb.co/v1Jpvh7/6-04.png>

<https://github.com/isidropo/EjerciciosC/blob/master/6.04.c>

```

        Metodo de division mediante restas sucesivas
a: 128
b: 7

Coeficiente: 18
```

Salida 21.- Problema 33

<https://i.ibb.co/5cj5kgV/6-04E.png>

6.05. Problema 34. Realizar.

Se quiere obtener el máximo común divisor de dos enteros introducidos por teclado, ahora bien, hay que hacer uso del **algoritmo de Euclides**.

El *algoritmo de Euclides* dice que dados dos enteros A y B ($A > B$), se resta B a A tantas veces como sea posible. Si la última resta es cero entonces B será el máximo común divisor, sino obtendremos un residuo C el cual se deberá restar a B y repetir el mismo proceso hasta que el resultado de la resta sea cero.

6.06. Problema 35. Realizar.

Escribir un programa que solicite del usuario un entero tipo *long* y a continuación muestre por pantalla los dígitos de las unidades, decenas, ... que contiene el número. Los dígitos deben ser mostrados tal cual se introduce el número y separados por un espacio.

6.07. Problema 36. Explicar.

Realice un programa que diga cuantas veces aparece un entero de un dígito (0 a 9) en otro entero tipo *long*. Estos dos enteros deben ser solicitados al usuario. Por ejemplo, el dígito 4 aparece en 4654 dos veces.

El tipo de dato para el numero introducido se va a coger tipo *long* para conseguir el mayor recorrido de un entero y a parte se va a advertir de que el número será de como máximo 9 dígitos. El tipo de dato para el dígito introducido y el contador será suficiente con tipo *short*.

El primer *for* sirve para saber de cuantos dígitos es el número al comprobar, obteniendo así su divisor correspondiente del orden de 10. El segundo *for* es el que obtiene cuantas veces se allá el dígito en el número, comprobando cada dígito del número con el dígito introducido.

```

1 #include <stdio.h>
2 void main()
3 {
4     short dig, cont=0;
5     long num, n=1, aux;
6     puts("\tApariciones de un dígito (0-9) en un número");
7     printf("Número(9 dígitos max):\t"); scanf("%d", &num);
8     printf("Dígito(1 dígito):\t"); scanf("%hd", &dig);
9
10    for(long i=num; i > 10; i = i/10)
11        n *= 10;
12
13    aux = num;
14    for(n; n>0; n= n/10)
15    {
16        if(aux/n == dig)
17            cont++;
18        aux = aux % n;
19    }
20    printf("\n %d aparece %d veces en %d", dig, cont, num);
21 }
```

Código 25.- Problema 36

<https://i.ibb.co/wRgPrBH/6-07.png>

<https://github.com/isidropo/EjerciciosC/blob/master/6.07.c>

```
Apariciones de un digito (0-9) en un numero
Número(9 digitos max): 14523151
Digito(1 digito):      1

1 aparece 3 veces en 14523151
```

Salida 22.- Problema 36

<https://i.ibb.co/YyDGKnm/6-07E.png>

6.08. Problema 37. Realizar.

Se quiere realizar un programa que pida al usuario un valor, si el valor se encuentra dentro de intervalo correspondiente, el programa termina, en otro caso se vuelve a pedir el valor. El intervalo será por ejemplo de 10 a 250.

6.09. Problema 38. Explicar.

Realiza un programa que terminará cuando dos valores *a* y *b* enteros positivos introducidos por teclado disten uno del otro 100 unidades.

Comprendiendo bien el enunciado y teniendo conocimiento sobre estructuras en bucle, este ejercicio se puede resolver fácilmente, es decir, mediante una estructura *do-while* en la que se pidan los datos y que se repita hasta cumplir una determinada condición. La condición será que *a* y *b* sean mayores que 0 y *b-a* sea mayor que 100.

```
1 #include <stdio.h>
2 void main()
3 {
4     long a, b, cond=0;
5     puts("Fin de programa -> a>0, b>0 y b-a>100");
6
7     do
8     {
9         printf("a: "); scanf("%d", &a);
10        printf("b: "); scanf("%d", &b);
11        if(a>0 && b>0 && b-a>100)
12            cond = -1;
13        else
14            printf("\n");
15    }while(cond != -1);
16 }
```

Código 26.- Problema 38

<https://i.ibb.co/nkHWW6P/6-09.png>



<https://github.com/isidropo/EjerciciosC/blob/master/6.09.c>

6.10. Problema 39. Realizar.

Crear un programa que solicite al usuario valores numéricos positivos y que una vez se introduzca un valor negativo se calcule la media de los valores introducidos. El último valor introducido no debe repercutir en el cálculo de la media.

6.11. Problema 40. Explicar.

Realizar un programa que muestre por pantalla todas las parejas par-impar con números de 0 a 10. La pareja (1, 2) se considera distinta de la pareja (2, 1).

Para la realización de este problema hay que hacer uso de dos estructuras en bucle *for* anidadas, una para iterar el número de la izquierda y otra para iterar el número de la derecha. Por último, se deberá comprobar que, si el número de la izquierda es par, el de la derecha deberá ser impar y viceversa, e imprimirlas.

```

1 #include <stdio.h>
2 void main()
3 {
4     for(int i=0; i<11; i++)
5         for(int j=0; j<11; j++)
6         {
7             if(i%2 == 0)
8             {
9                 if(j%2 > 0)
10                     printf("(%d, %2d)\n", i, j);
11             }
12             else if(i%2 > 0)
13             {
14                 if(j%2 == 0)
15                     printf("\t(%d, %2d)\n", i, j);
16             }
17         }
18 }
```

Código 27.- Problema 40

<https://i.ibb.co/f4PBk01/6-11.png>

<https://github.com/isidropo/EjerciciosC/blob/master/6.11.c>



6.12. Problema 41. Desbloquear.

Se necesita saber cuántas veces se cumple el teorema de Pitágoras ($a^2 + b^2 = c^2$) para los enteros de 1 a 50. Realiza un programa que cumpla con dicho cometido.

¿Cuál es el bucle más eficaz para desarrollar este ejercicio?

for V, while, do-while

¿Cuántos bucles habrá que utilizar?

1, 2, 3 V

¿De qué forma habrá que utilizar los bucles?

Secuencia, Anidamiento V

3/3 1S 2S 3S ¡Muy bien! Con la ayuda de tres bucles *for* anidados y en el más interno introduciendo la condición del teorema de Pitágoras se conseguirá resolver el problema.

2/3 1N 2S 3S ¡Regular! El problema se puede resolver con bucles *while* y *do-while* pero la manera más optima es hacerlo con tres bucles *for* anidados.

2/3 1S 2N 1S ¡Regular! No se puede resolver el problema con 1 o 2 estructuras en bucle, es necesario iterar las tres variables del teorema de Pitágoras. La manera más optima es hacerlo con tres bucles *for* anidados.

2/3 1S 2S 3N ¡Regular! No se puede resolver el problema mediante bucles en secuencia porque se perdería la dependencia de cada variable con las demás. La manera más optima es hacerlo con tres bucles *for* anidados.

1/3 0/3 ¡Imposible! Se necesita como mínimo mantener una dependencia entre las variables para recorrer todos los resultados y mostrar los que cumplen la condición.

```

1 #include <stdio.h>
2 void main()
3 {
4     for(long a=1; a<51; a++)
5         for(long b=1; b<51; b++)
6             for(long c=1; c<51; c++)
7                 if(a*a + b*b == c*c)
8                     printf ("%d, %d, %d)\n", a, b, c);
9 }
```

Código 28.- Problema 41

<https://i.ibb.co/HBwjWSS/6-12.png>

<https://github.com/isidropa/EjerciciosC/blob/master/6.12.c>

6.13. Problema 42. Realizar.

Se pide realizar un programa que muestre por pantalla el valor factorial de los enteros de 0 a 10. Imprime los valores del siguiente modo ($3! = 3 \times 2 \times 1 = 6$).

6.14. Problema 43. Realizar

Crear un programa que muestre por pantallas las tablas de multiplicar del 1 al 10. La presentación debe ser lo más legible posible incluyendo en la función `printf()` el carácter tabulador (`\t`), alineador de numero (`%nd`) y carácter nueva línea (`\n`) adecuados.

Nota: El alineador de números sirve para imprimir números de manera alineada, como si de una columna se tratase. Por ejemplo, la instrucción `printf("%3d", a)` imprimirá tres huecos y después `a` o lo que es lo mismo, imprimirá `a` con tres huecos a su izquierda. Si el número tuviese de 3 dígitos en adelante no se desplazaría ningún hueco, sin embargo, si tuviese menos de 3 se desplazaría la cantidad de huecos restantes, es decir, para dos dígitos se desplazaría un hueco y para un digito se desplazaría dos huecos.

6.15. Problema 44. Desbloquear.

Realiza un programa que calcule la suma de números primos introducidos por teclado. Una vez se introduzca un número no primo se mostrará la suma y finalizará el programa.

¿Cuál sería la estructura en bucle más aconsejable para realizar esta tarea?

while V, do-while V, for

Tanto `while` como `do-while` serían las mejores opciones, aunque hay que decir que **`do-while`** sería la que menos líneas de código aportaría. ¿Por qué NO usar una estructura de repetición `for`? Pues por el simple hecho de que no es necesario aumentar un contador cada turno, solo se necesita un variable que, bajo unas condiciones, rompa el bucle y finalice el programa.

¿Por qué NO usar una estructura de repetición `for`? Pues por el simple hecho de que no es necesario aumentar un contador cada turno. Solo se necesita un variable que, bajo unas condiciones, rompa el bucle y finalice el programa.

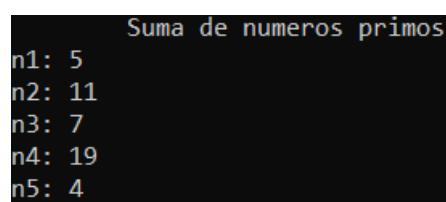
```

1 #include <stdio.h>
2 void main()
3 {
4     short trigger=1, cont=0;
5     long num, suma=0;
6     puts("\tSuma de numeros primos");
7
8     do
9     {
10         printf("n%d: ", trigger); scanf("%d", &num);
11         for(long i=1; i<=num; i++)
12             if(num%i == 0)
13                 cont++;
14         if(cont > 2)
15         {
16             printf("La suma es %d", suma);
17             trigger = -1;
18         }
19     else
20     {
21         suma += num;
22         trigger++;
23         cont = 0;
24     }
25 }while(trigger != -1);
26 }
```

Código 29.- Problema 44

<https://i.ibb.co/Sv79nmn/6-15.png>

<https://github.com/isidropo/EjerciciosC/blob/master/6.15.c>



```

Suma de numeros primos
n1: 5
n2: 11
n3: 7
n4: 19
n5: 4
```

Salida 23.- Problema 44

<https://i.ibb.co/dj0LR6j/6-15E.png>



Capítulo 7.

7.01. Problema 45. Desbloquear.

Realizar un programa que calcule el *máximo común divisor* de dos números introducidos por consola. El proceso se debe repetir hasta que el usuario introduzca un par de ceros (haz uso de la palabra reservada *break*).

Haz uso de una estructura en bucle para calcular el *mcd*. Se requiere saber cuántas veces se repite dicho bucle mediante el uso de una variable definida DENTRO de él, ¿Qué especificador es el más adecuado?

extern, static V, auto

¡Por supuesto! Una variable con especificador *static* conserva su valor, aunque finalice el bloque donde actúa, eso sí, no será visible fuera de él ni para lectura ni para escritura.

¡No! Se recomienda repasar los especificadores *static*, *extern* y *auto* explicados en el *Capítulo 7*.

```
1 #include <stdio.h>
2 void main()
3 {
4     long a, b, mcd;
5     do
6     {
7         printf("a: "); scanf("%d", &a);
8         printf("b: "); scanf("%d", &b);
9         if(a == 0 && b == 0)
10             break;
11         while(b)
12         {
13             static short cont = 0;
14             mcd = b;
15             b = a%b;
16             a = mcd;
17             printf("\ncont = %hd", ++cont);
18         }
19         printf("\nEl mcd es %d.\n\n", mcd);
20     }while(1);
21 }
```

Código 30.- Problema 45

<https://i.ibb.co/JKg7KWd/7-01.png>

<https://github.com/isidropo/EjerciciosC/blob/master/7.01.c>

```
a: 12
b: 5

cont = 1
cont = 2
cont = 3
El mcd es 1.

a: 16
b: 2

cont = 4
El mcd es 2.

a: 15
b: 3

cont = 5
El mcd es 3.

a: 0
b: 0
```

Salida 24.- Problema 45

<https://i.ibb.co/DGcbfDX/7-01E.png>

7.02. Problema 46. Realizar.

Se quiere implementar un programa que pida al usuario dos enteros positivos y que a continuación se calcule su **máximo común divisor** y su **mínimo común múltiplo**.

El cálculo del mcd se debe hacer con una función que acepte dos parámetros *a* y *b*, y mcm con otra función que acepte un parámetro (*mcd*) y utilice la siguiente ecuación: $a \times b = mcd(a, b) \times mcm(a, b)$.

7.03. Problema 47. Explicar.

Es preciso realizar una función para el cálculo del número de Fibonacci. La función debe aceptar un valor *n* del término *n*-ésimo que se quiere calcular. Es preciso seguir el siguiente prototipo: *long Fibonacci (long);*

Tanto el valor aceptado como el devuelto por la función van a ser de tipo *long*, ya que la serie utiliza y calcula solo números enteros.

Las buenas prácticas de programación, en cuanto a funciones se refiere, nos dicen que es recomendable poner los prototipos de las funciones debajo de las macros (#) para aumentar la claridad del programa. Por lo tanto, en primer lugar, pondremos las macros, en segundo lugar, pondremos los prototipos de las funciones, a continuación, situaremos el *main* y en última instancia, definiremos las funciones.

Por último, decir que los dos primeros términos de la serie de Fibonacci son constantes y se devolverán tal cual.

```

1 #include <stdio.h>
2 long fibonacci(long);
3 void main()
4 {
5     long num;
6     printf("Termino n de la serie de Fibonacci: ");
7     scanf("%d", &num);
8
9     printf("n->%d: %d", num, fibonacci(num));
10 }
11
12 long fibonacci(long n)
13 {
14     long ant=0, act=1, aux;
15     if(n == 0)      return 0;
16     else if(n == 1) return 1;
17     else
18     {
19         for(int i=1; i<n; i++)
20         {
21             aux = ant;
22             ant = act;
23             act = act + aux;
24         }
25         return act;
26     }
27 }
```

Código 31.- Problema 47

<https://i.ibb.co/CvmNxrb/7-03.png>

<https://github.com/isidropo/EjerciciosC/blob/master/7.03.c>

Termino n de la serie de Fibonacci: 16
n->16: 987

Salida 25.- Problema 47

<https://i.ibb.co/rpJHj8Q/7-03E.png>



7.04. Problema 48. Explicar.

Se debe crear una función para verificar si el número proporcionado es o no perfecto, se devolverá 1 si lo es y 0 si no lo es. Ahora, haga uso de la función para mostrar por pantalla los números perfectos comprendidos en un intervalo proporcionado por teclado.

Nota: Un número perfecto es un número natural positivo que es igual a la suma de sus divisores propios. Los primeros números perfectos son 6, 28, 496 y 8128.

Se va a hacer uso de las constantes enumeradas para construir un nuevo tipo *bool* que devolverá True (= 1) o False (= 0). El tipo *bool* será el tipo de retorno de la función, *esPerfecto*, que se encarga de decir si un número dado es perfecto o no.

Una vez se tiene la función tan solo hay que pedir los datos al usuario e iterarlos en un bucle *for*.

Dado que los números perfectos distan mucha cantidad unos de otros se utilizarán para todas las variables el tipo de dato entero más grande posible (*long*). Hay que decir que a partir del cuarto número perfecto el programa tarda demasiado en calcular el siguiente.

```

1 #include <stdio.h>
2 //Declaracion del tipo bool manualmente
3 enum bool { False, True };
4 enum bool esPerfecto(long);
5 void main()
6 {
7     long a, b;
8     puts("Numeros perfectos en el intervalo [a, b]");
9     printf("a: "); scanf("%d", &a);
10    printf("b: "); scanf("%d", &b);
11    for(long i=a; a<b; a++)
12        if(esPerfecto(a))
13            printf("%d\n", a);
14 }
15
16 enum bool esPerfecto(long n)
17 {
18     long suma=0;
19     enum bool bool1;
20     for(long i=1; i<n; i++)
21         if(n%i == 0)
22             suma += i;
23     if(suma == n)
24         return bool1 = True;
25     else
26         return bool1 = False;
27 }

```

Código 32.- Problema 48

<https://i.ibb.co/WvZrNk3/7-04.png>

<https://github.com/isidropo/EjerciciosC/blob/master/7.04.c>

```

Numeros perfectos en el intervalo [a, b]
a: 1
b: 1000
6
28
496

```

Salida 26.- Problema 48



<https://i.ibb.co/phZ657j/7-04E.png>

7.05. Problema 49. Realizar.

$$(x + y)^n = \frac{n!}{k! * (n - k)!} * x^{n-k} * y^k$$

Se quiere dar solución al binomio de Newton,

$$(x + y)^n = \frac{n!}{k! * (n - k)!} * x^{n-k} * y^k$$

<https://i.ibb.co/Yt1CWDZ/7-05-Formula.png>

para k entre $[0, n]$. Pedir al usuario los valores de x, y, n y crear una función que se encargue de resolver el binomio y devolver un resultado. Resultará útil crear otra función que calcule el factorial de cualquier número.

Además, se debe utilizar la función con prototipo `double pow(double base, double exp);` para calcular potencias. Dicha función está incluida en `<math.h>`.

7.06. Problema 50. Explicar.

Un **número primo perfecto** es un número n que es primo a la vez que $(n-1)/2$ también lo es. Por ejemplo, el 11. Entonces, se requiere crear un programa que calcule los primos perfectos menores que 1000.

Después de estudiar detenidamente las posibles opciones para la realización del ejercicio, se ha llegado a la conclusión de que, creando una función que verifique si un número es primo es más que suficiente. La función recorre todos los valores entre 1 y n (número pasado como parámetro) y si n tiene exactamente dos divisores entonces será primo.

La función `main` imprime todos los números menores que 1000 que verifiquen ser primos perfectos, es decir, que tanto n como $(n-1)/2$ sean primos.

```

1 #include <stdio.h>
2 long esPrimo(long);
3 void main()
4 {
5     puts("\tNumeros primos perfectos menores que 1000\n");
6     for(long n=1; n<1000; n++)
7         if(esPrimo(n) && esPrimo((n-1)/2))
8             printf("%3d\t", n);
9 }
10
11 long esPrimo(long n)
12 {
13     long cont=0;
14     for(long i=1; i<=n; i++)
15         if(n%i == 0) cont++;
16     if(cont == 2) return 1;
17     else return 0;
18 }
```

Código 33.- Problema 50

<https://i.ibb.co/dm211KT/7-06.png>

<https://github.com/isidrop/C/blob/master/7.06.c>

Numeros primos perfectos menores que 1000														
5	7	11	23	47	59	83	107	167	179	227	263	347	359	383

Salida 27.- Problema 50

<https://i.ibb.co/VtMLxhj/7-06E.png>

7.07. Problema 51. Explicar.

Realizar una función que, aceptando un número, muestre por pantalla su factorización con los factores comunes agrupados (en forma de potencia).

Nota: El uso de vectores haría este programa mucho más sencillo, pero aún no se ha visto como trabajar con vectores (En primer lugar, se utilizaría un bucle *for* para guardar cada factor junto a su cantidad en un vector *y*, en segundo lugar, se utilizaría otro bucle *for* para mostrar los datos por pantalla).

Una vez dicho esto hay que buscar algún método de conseguir el resultado mediante estructuras de selección, repetición, y algunas variables.

Para comenzar la función, si el número pasado es menor que 1 se mostrará un error, si es igual a 1 se imprimirá tal cual y si es mayor se realizará un bucle *for*. El bucle irá comprobando los



factores... en caso de repetirse aumentará un contador de factor común, en caso contrario imprimirá el factor actual y se continuará con el siguiente.

```

1 #include <stdio.h>
2 void factorizar (long n) {
3     long cont=0;
4     if(n <= 0) {
5         printf ("\xAD Error! %ld no se puede factorizar", n);
6         return;
7     }
8     else if(n == 1) { printf("1\n"); return; }
9     for(long i=2; i<=n; i++) {
10         cont=0;
11         while (n%i == 0) {
12             ++cont;
13             n/=i;
14         }
15         if (cont>0)
16             printf ("%ld^%d\n", i, cont);
17     }
18 }
19
20 void main()
21 {
22     long num;
23     printf ("Factorizacion del numero... ");
24     scanf ("%ld", &num);
25
26     factorizar(num);
27 }
```

Código 34.- Problema 51

<https://i.ibb.co/zVMy2ZM/7-07.png>

<https://github.com/isidrop/C/blob/master/7.07.c>

```
Factorizacion del numero... 345
3^1
5^1
23^1
```

Salida 28.- Problema 51

<https://i.ibb.co/m0pKksT/7-07E.png>

Capítulo 8.

8.01. Problema 52. Explicar.

Se pide realizar un programa que declare un array de 100 enteros y que a continuación se le asignen los cien primeros números impares.

En primer lugar, se declarará un array de 100 números enteros. Ahora, para asignar valores en serie a cada índice del array, primero se necesita de un bucle que mueva dicho índice (*for* es el mejor candidato) y entonces se le asignarán los valores a cada índice. Por último, para mostrar los valores por pantalla es necesario otro bucle que mueva de nuevo el índice del array.

```

1 #include <stdio.h>
2 void main()
3 {
4     long impares[100], cont=1;
5     puts ("\t100 primeros impares positivos");
6     for(long i=0; i<100; i++)
7     {
8         impares[i] = cont;
9         cont += 2;
10    }
11    for(long i=0; i<100; i++)
12        printf ("%3d\t", impares[i]);
13 }
```

Código 35.- Problema 52

<https://i.ibb.co/MGrYfrN/8-01.png>

<https://github.com/isidropa/EjerciciosC/blob/master/8.01.c>

100 primeros impares positivos														
1	3	5	7	9	11	13	15	17	19	21	23	25	27	29
31	33	35	37	39	41	43	45	47	49	51	53	55	57	59
61	63	65	67	69	71	73	75	77	79	81	83	85	87	89
91	93	95	97	99	101	103	105	107	109	111	113	115	117	119
121	123	125	127	129	131	133	135	137	139	141	143	145	147	149
151	153	155	157	159	161	163	165	167	169	171	173	175	177	179
181	183	185	187	189	191	193	195	197	199					

Salida 29.- Problema 52

<https://i.ibb.co/ySbnH2g/8-01E.png>

8.02. Problema 53. Realizar

Realizar un programa que almacene 10 elementos por teclado en un vector y después copie esos diez elementos en otro, pero de forma inversa.

8.03. Problema 54. Realizar

Realizar un programa que declare tres vectores (arrays) de 100 elementos. Al primero de ellos se le tiene que introducir valores aleatorios entre 0 y 300, al segundo se le tienen que introducir los valores pares y al tercero los valores impares.

Nota: La función `rand()` devuelve números pseudoaleatorios entre 0 y 32767, ahora mediante el operador resto, `rand() % num+1`, se pueden conseguir valores aleatorios entre 0 y `num`. La función es pseudoaleatoria ya que siempre generará las mismas secuencias. Si se quisiera solventar este problema se ha de usar la función `srand(x)` que cambia las secuencias aleatorias, donde `x` deberá ser un valor que cambie con el tiempo como la función `time(NULL)`, `srand(time(NULL))`.

8.04. Problema 55. Explicar.

Se desea realizar un programa que mediante un vector (array) aleatorio de 100 valores de 0 a 100, se encuentre el mayor de ellos y además donde está ubicado (índice). En caso de múltiples apariciones, es suficiente con indicar la primera.

Nota: La función `rand()` devuelve números pseudoaleatorios entre 0 y 32767, ahora mediante el operador resto, `rand() % num+1`, se pueden conseguir valores aleatorios entre 0 y `num`. La función es pseudoaleatoria ya que siempre generará las mismas secuencias. Si se quisiera solventar este problema se ha de usar la función `srand(x)` que cambia las secuencias aleatorias, donde `x` deberá ser un valor que cambie con el tiempo como la función `time(NULL)`, `srand(time(NULL))`.

Para este problema solo se ha creado una función que se encarga de buscar y devolver el índice donde está el mayor valor del vector. Por lo tanto, sabemos el índice del mayor y su valor. Esta función acepta dos parámetros: por un lado, el vector y por otro, su tamaño, de esta forma se consigue una función versátil para cualquier vector.

En la función `main` se creará el vector aleatorio mediante un bucle `for` y la función `rand()`. Al final se mostrará tanto la posición como la cantidad del valor mayor.

Nota: La función `rand()` devuelve números pseudoaleatorios entre 0 y 32767, ahora mediante el operador resto, `rand() % num+1`, se pueden conseguir valores aleatorios entre 0 y `num`. La función es pseudoaleatoria ya que siempre generará las mismas secuencias. Si se quisiera solventar este problema se ha de usar la función `srand(x)` que cambia las secuencias aleatorias, donde `x` deberá ser un valor que cambie con el tiempo como la función `time(NULL)`, `srand(time(NULL))`.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 long ind_mayor(const long, long v[]);
4 void main()
5 {
6     long aleatorios[100], pos;
7     srand(time(NULL));
8     puts("\t100 numeros aleatorios entre 0 y 100:\n");
9
10    for(long i=0; i<100; i++)
11    {
12        aleatorios[i] = rand() % 101;
13        printf("%3d\t", aleatorios[i]);
14    }
15    pos = ind_mayor(100, aleatorios);
16    printf("\n\nv[%d]: %d", pos, aleatorios[pos]);
17 }
18
19 long ind_mayor(const long n, long v[n])
20 {
21     long mayor = v[0], indice=0;
22     for(long i=1; i<n; i++)
23         if(mayor < v[i]) { mayor = v[i]; indice = i; }
24     return indice;
25 }
```

Código 36.- Problema 55

<https://i.ibb.co/jM4zKT0/8-04.png>

<https://github.com/isidropo/EjerciciosC/blob/master/8.04.c>

100 numeros aleatorios entre 0 y 100:															
88	3	96	7	35	95	82	37	15	88	14	74	63	17	69	
17	67	97	32	18	86	87	17	7	77	27	83	75	91	14	
39	1	32	97	1	93	24	67	93	92	26	74	27	79	80	
24	99	99	40	64	37	70	29	4	14	17	59	84	73	9	
7	1	71	59	7	91	51	93	66	68	49	16	97	100	91	
20	19	47	26	64	72	2	70	72	34	8	28	89	39	55	
48	100	48	6	23	36	56	0	72	57						
v[73]: 100															

Salida 30.- Problema 55

<https://i.ibb.co/P1npBRh/8-04E.png>

Aviso: Si el prototipo de la función incluye un array como parámetro, este ha de ser escrito del mismo modo que en la definición de la función (C debe saber si el parámetro se trata de un array o de una variable).

8.05. Problema 56. Realizar

Realizar un programa que cree un vector (array) aleatorio, de tamaño introducido por teclado, mediante una función, la función debe aceptar dos parámetros: el vector y su tamaño. Una vez hecho esto se pide ordenar el vector con el *algoritmo de burbuja* (visto en teoría del Capítulo 8) creando las funciones necesarias.

Aviso: Según el *estándar C99*, para crear un array con un tamaño introducido por teclado hay que pedir el tamaño primero e introducírselo *jen la definición!*

8.06. Problema 57. Realizar.

Se quiere realizar un grupo de funciones reutilizables que trabajen con vectores (arrays), cuyos prototipos son los siguientes:

short sonPares (const long n, long v[n]);

short buscarValor (const long n, long v[n], long valor);

short estanOrdenados (const long n, long v[n]);

-sonPares(): Los vectores admitidos serán tipo *long*, así como sus tamaños. La función devolverá 1 si todos sus valores son pares o 0 si hay alguno que no sea par.

-buscarValor(): Los vectores admitidos serán tipo *long*, así como sus tamaños y el valor a buscar. La función devolverá 1 si se ha encontrado el valor buscado o 0 si no.

-estanOrdenados(): Los vectores admitidos serán tipo *long*, así como sus tamaños. La función devolverá 1 si los valores están ordenados creciente o decrecientemente o 0 si no.

8.07. Problema 58. Desbloquear

Crear una función que determine si en un array hay algún valor que no sea primo, es decir, devolverá 0 si hay al menos un valor no primo o 1 si todos los valores son primos.

En este caso, ¿qué es más aconsejable? ¿una función que verifique si hay algún valor no primo? o ¿dos funciones, una que verifique algún valor no primo y otra que verifique si un número es primo?

1 función, 2 funciones V

¡Exacto! Con el uso de dos funciones conseguimos un código más sencillo y reutilizable. Cada una de las funciones puede ser usada en cualquier momento prescindiendo una de la otra.

Aunque se puede conseguir el funcionamiento deseado con una sola función, es más aconsejable separar cada tarea en funciones distintas. De esta forma se consigue el mismo



resultado, pero se dispone de ambas funciones para, por ejemplo, utilizar la función *primo()* en otra parte del código.

```

1  #include <stdio.h>
2  short primo(long n)
3  {
4      short c=0;
5      for(long i=1; i<=n; i++)
6          if(n%i == 0)
7              c++;
8      if(c == 2)  return 1;
9      else        return 0;
10 }
11 short todosPrimos(long v[], long tam)
12 {
13     short c=0;
14     for(long i=0; i<tam; i++)
15         if(primo(v[i]))
16             c++;
17     if(c == tam) return 1;
18     else        return 0;
19 }
20
21 void main()
22 {
23     long v1[3] = {3, 5, 11}, v2[3] = {1, 2, 3};
24     puts("\t1=Todos son primos\t0=Alguno no es primo");
25     printf("v1[]: %d, %d, %d\n", v1[0], v1[1], v1[2]);
26     printf("v2[]: %d, %d, %d\n", v2[0], v2[1], v2[2]);
27     printf("v1[] -> %hd\n", todosPrimos(v1, 3));
28     printf("v2[] -> %hd", todosPrimos(v2, 3));
29 }
```

Código 37.- Problema 58

<https://i.ibb.co/F0ybZ9V/8-07.png>

<https://github.com/isidrop/C/blob/master/8.07.c>

1=Todos son primos	0=Alguno no es primo
v1[]: 3, 5, 11	
v2[]: 1, 2, 3	
v1[] -> 1	
v2[] -> 0	

Salida 31.- Problema 58

<https://i.ibb.co/S0s1R05/8-07E.png>

Aviso: En el paso de vectores (arrays) a funciones, si se pasa **v** se está pasando la dirección al primer elemento (paso por referencia), es decir, el vector, y si se escribe **v[num]** se está pasando el valor (paso por copia) de la posición *num*.

8.08. Problema 59. Desbloquear

Se quieren rotar los valores de un vector a la derecha un número determinado de posiciones mediante una función. La rotación implica que el valor extremo derecho pasará al valor extremo izquierdo.

¿Cuántos parámetros debe aceptar la función?

2, 3 V

¡Eso es! Se necesita como mínimo: *el vector* a modificar, *el tamaño del vector* para cambiar todos los valores de posición y además de saber dónde termina el vector, y la *cantidad de rotaciones*. Además, se puede crear una función, *intercambio()*, que se encargue de intercambiar dos números mediante el paso por referencia para modular más el código.

¡Imposible! No existe la forma de conseguir la rotación de un vector mediante una función que acepte solamente dos parámetros. Como mínimo se necesita *el vector* a modificar, *el tamaño del vector* para cambiar todos los valores de posición y además de saber dónde termina el vector, y la *cantidad de rotaciones*.

```

1 #include <stdio.h>
2 void rotarVectorDer(const int tam, int v[tam], int nRot)
3 {
4     int aux;
5     for(int j=1; j<=nRot; j++)
6         for(int i=0; i<tam; i++)
7         {
8             aux = v[i];
9             v[i] = v[tam-1];
10            v[tam-1] = aux;
11        }
12    }
13
14 void main()
15 {
16     int vector[5] = {0, 1, 2, 3, 4};
17     printf("Vector: ");
18     for(short i=0; i<5; i++)
19         printf("%d ", vector[i]);
20
21     printf("\nVector rotado 2 posiciones: ");
22     rotarVectorDer(5, vector, 2);
23     for(short i=0; i<5; i++)
24         printf("%d ", vector[i]);
25 }
```

Código 38.- Problema 59

<https://i.ibb.co/WnptrkF/8-08.png>

<https://github.com/isidropo/EjerciciosC/blob/master/8.08.c>

```
Vector: 0 1 2 3 4
Vector rotado 2 posiciones: 3 4 0 1 2
```

Salida 32.- Problema 59

<https://i.ibb.co/S3FmY4J/8-08E.png>

8.09. Problema 60. Realizar

Realizar un programa que calcule el histograma de un vector (array) de 10000 enteros con valores aleatorios de 0 a 99. Para la realización del histograma se requiere otro vector que almacene la cantidad de cada valor en una de sus 100 posiciones. Una vez almacenados los

10000 enteros en el vector histograma se debe mostrar por pantalla cada número junto con su cantidad.

Nota: La función `rand()` devuelve números pseudoaleatorios entre 0 y 32767, ahora mediante el operador resto, `rand() % num+1`, se pueden conseguir valores aleatorios entre 0 y `num`. La función es pseudoaleatoria ya que siempre generará las mismas secuencias. Si se quisiera solventar este problema se ha de usar la función `srand(x)` que cambia las secuencias aleatorias, donde `x` deberá ser un valor que cambie con el tiempo como la función `time(NULL)`, `srand(time(NULL))`.

8.10. Problema 61. Desbloquear.

Realizar el código de un programa que, mediante un array de 20 valores aleatorios entre [1, 25], determine cuales de esos valores están repetidos y cuantas veces esta repetido cada uno.

¿Cuántos arrays se necesitarán para cumplir el cometido del programa?

1, 2 V, 3 V, 4

¡Correcto! Tanto **2** como **3** se consideran opciones válidas, eso sí en el caso de utilizar 2; una ha de ser de dos dimensiones: una dimensión para almacenar el valor y otra para la cantidad. La opción de **3** arrays es más sencilla de implementar, en contraposición la opción de **2** arrays genera un código más corto y entendible.

¡No! Mediante un solo array nunca se podrá conseguir... se necesita almacenar cada valor distinto en otro array para saber si hay más de uno.

¡No! Utilizar más de tres arrays es innecesario, quizás se puedan utilizar, pero resulta en un aumento de memoria utilizada, aumento de código y mayor complejidad.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 void main()
4 {
5     srand(time(NULL));
6     int si=0, c=0, v[20], v2[20][2]={0};
7
8     printf("\tAgrupador de valores aleatorios\nValores: ");
9     for(int i=0; i<20; i++){
10         v[i] = rand() % 25 + 1;
11         printf("%d ", v[i]);
12     }
13     for(int i=0; i<20; i++){
14         for(int j=0; j<20; j++){
15             if(v2[j][0] == v[i]){
16                 v2[j][1]++;
17                 si = 1;
18             }
19             if(si)
20                 si=0;
21         else{
22             v2[c][0] = v[i];
23             v2[c][1]++;
24             c++;
25         }
26     }
27     printf("\nValores repetidos: ");
28     for(int i=0; v2[i][0] > 0; i++)
29         if(v2[i][1] > 1)
30             printf("\n %2d -> %2d veces", v2[i][0], v2[i][1]);
31 }

```

Código 39.- Problema 61

<https://i.ibb.co/Nymftbx/8-10.png>

<https://github.com/isidropo/EjerciciosC/blob/master/8.10.c>

```

Agrupador de valores aleatorios
Valores: 23 15 5 4 24 8 18 9 18 19 14 24 25 12 23 15 24 20 9 8
Valores repetidos:
 23 -> 2 veces
 15 -> 2 veces
 24 -> 3 veces
 8 -> 2 veces
 18 -> 2 veces
 9 -> 2 veces

```

Salida 33.- Problema 61

<https://i.ibb.co/gWtXQ3C/8-10E.png>

8.11. Problema 62. Realizar.

Realizar un programa que almacene en un array las posiciones de un polinomio de máximo 2º grado desde la posición 0 (termino independiente) hasta la posición 2 (termino de x^2). Una vez obtenido el polinomio por teclado deberá ser resultado, almacenando en otro array los diferentes resultados. Entonces, el primer array deberá tener 3 elementos como máximo y el segundo 2.

Adicionalmente se quiere realizar una función que realice la suma de dos polinomios de máximo grado 4, para ello se deben pasar los dos arrays suma y el array solución. El tamaño de los arrays será siempre 4 independientemente de los términos que incluyan. El prototipo de la función será el siguiente:

```
void sumaP(long v1[], long v2[], long vSol[]);
```

Nota: Los arrays son pasados a las funciones **por referencia**, es decir, si se modifican en la función también se modificarán en el programa principal.

8.12. Problema 63. Realizar.

Realizar el código de un programa que calcule la resta de dos arrays bidimensionales. Ambos arrays deben ser pedidos por teclado. El resultado debe ser almacenado en otro array bidimensional y posteriormente mostrado por pantalla en forma de matriz.

8.13. Problema 64. Desbloquear.

Teniendo una matriz (array bidimensional) se requiere: en primer lugar, copiar todos los valores pares uno tras otro de esa matriz en otra y, a continuación, copiar todos los valores impares restantes.

Ambas matrices tendrán el mismo número de filas que de columnas. Por ejemplo, para la matriz:

1 2 3

3 6 8

5 8 7



Se creará otra:

2 6 8

8 1 3

3 5 7

¿Se puede realizar el ejercicio mediante **2** bucles *for* solamente?

Sí, No V

¡No es correcto! Es necesario para cada acción que implique recorrer la matriz completa, dos bucles *for* anidados. Por lo tanto, para analizar todos los valores pares y luego los valores impares hay que utilizar dos parejas de bucles *for* anidados.

Existen otras formas de conseguir el cometido como guardando los valores impares en un array mientras analizamos los pares, pero aun así se necesitarán más de dos bucles *for*. ¡Te animo a que intentes realizarlo para comprender lo que se ha comentado aquí!

¡Exacto! Si se utilizasen dos bucles *for*, en este caso anidados, solo se podrían analizar los valores pares. Ahora, siguiendo las mismas pautas se utilizarían otros dos para analizar los impares.

Existen otras formas de conseguir el cometido como guardando los valores impares en un array mientras analizamos los pares. Pero aun así se necesitaría como mínimo otro bucle *for* para incluirlos en la matriz, es decir, tres en total.

```

1 #include <stdio.h>
2 void main()
3 {
4     long m2[3][4], m1[3][4] = { {1, 2, 7, 7},
5                                 {8, 4, 5, 5},
6                                 {6, 3, 8, 6} };
7     long cF=0, cC=0;
8     //Introducir valores pares de m1 en m2
9     for(long i=0; i<3; i++)
10        for(long j=0; j<4; j++)
11            if(m1[i][j] %2 == 0)
12            {
13                m2[cF][cC++] = m1[i][j];
14                if(cC > 3)
15                {
16                    cF++;
17                    cC=0;
18                }
19            }
20     //Introducir valores impares de m1 en m2
21     for(long i=0; i<3; i++)
22        for(long j=0; j<4; j++)
23            if(m1[i][j] %2 > 0)
24            {
25                m2[cF][cC++] = m1[i][j];
26                if(cC > 3)
27                {
28                    cF++;
29                    cC=0;
30                }
31            }
32     //Imprimir m2
33     for(long i=0; i<3; i++)
34     {
35         for(long j=0; j<4; j++)
36             printf(" %d", m2[i][j]);
37         printf("\n");
38     }

```

Código 40.- Problema 64

<https://i.ibb.co/5chdPsm/8-13.png>

<https://github.com/isidrop/C/blob/master/8.13.c>



2	8	4	6
8	6	1	7
7	5	5	3

Salida 34.- Problema 64

<https://i.ibb.co/wzrCjbS/8-13E.png>

8.14. Problema 65. Explicar.

Realizar un programa que mediante una matriz 3x4 modifique, si es preciso, dicha matriz para que la suma de los valores de su periferia supere a cualquier otro valor de la matriz.

Después de pensar detenidamente sobre el problema se llega a la conclusión que los valores que se deben comprobar son solamente los que encierra la periferia de la matriz.

A partir de aquí, se compara la suma de los valores de la periferia con cualquiera de los valores internos de la matriz. Si uno de los valores supera a la suma de la periferia, será sustituido por un valor de la periferia. Cuando termine la comparación se mostrará la matriz (modificada o sin modificar).

Para reducir el código se crearán dos funciones: una que imprimirá la matriz por pantalla y otra que calculará la suma de los valores de la periferia.

```

1 #include <stdio.h>
2 int sumaValPeriMatriz(int v[3][4], int n, int m)
3 {
4     int i, j, suma=0;
5     i=0;    for(j=0; j<m; j++)
6             suma += v[i][j];
7     i=n-1;  for(j=0; j<m; j++)
8             suma += v[i][j];
9     j=0;    for(i=1; i<n-1; i++)
10            suma += v[i][j];
11     j=m-1; for(i=1; i<n-1; i++)
12            suma += v[i][j];
13     return suma;
14 }
15 void impMatriz(int v[3][4], int n, int m)
16 {
17     for(int i=0; i<n; i++)
18     {
19         for(int j=0; j<m; j++)
20             printf(" %2d", v[i][j]);
21         puts("");
22     }
23 }
```

Código 41.- Problema 65 - 1

<https://i.ibb.co/6PXYw8C/8-14-1.png>

```

24 void main()
25 {
26     printf ("\tLa suma de los valores de la periferia debe ");
27     printf ("superar a cualquier otro valor de la matriz");
28     int n=3, m=4;
29     int periferia, aux, matriz[3][4] = { {1, 3, 6, 1},
30                                         {2, 41, 80, 6},
31                                         {2, 7, 9, 3} };
32     //Matriz
33     printf ("\nMatriz 3x4:\n\n");
34     impMatriz(matriz, n, m);
35     periferia = sumaValPeriMatriz(matriz, n, m);
36     printf ("Suma periferia: %d\n", periferia);
37     //Comprobar y modificar
38     for(int i=1; i<n-1; i++)
39         for(int j=1; j<m-1; j++)
40             if(matriz[i][j] > periferia)
41             {
42                 aux = matriz[i][j];
43                 matriz[i][j] = matriz[0][j];
44                 matriz[0][j] = aux;
45                 periferia = sumaValPeriMatriz(matriz, n, m);
46                 j--;
47             }
48     //Matriz modificada
49     printf ("\nMatriz 3x4 (Modificada):\n\n");
50     impMatriz(matriz, n, m);
51     printf ("Suma periferia: %d\n", periferia);
52 }
```

Código 42.- Problema 65 - 2

<https://i.ibb.co/jMy1XCr/8-14-2.png>

<https://github.com/isidropo/EjerciciosC/blob/master/8.14.c>



La suma de los valores de la periferia debe superar a cualquier otro valor de la matriz
Matriz 3x4:

```
1 3 6 1
2 41 80 6
2 7 9 3
Suma periferia: 40
```

Matriz 3x4 (Modificada):

```
1 41 80 1
2 3 6 6
2 7 9 3
Suma periferia: 152
```

Salida 35.- Problema 65

<https://i.ibb.co/6BLFgq8/8-14E.png>

Capítulo 9.

9.01. Problema 66. Explicar.

Mediante una cadena de texto introducida por teclado se ha de imprimir cuantas veces aparece cada una de las cinco vocales. Utiliza la sentencia de selección *switch* para saber si cada carácter es o no una de las vocales.

Las cadenas de texto en C solo se pueden almacenar en arrays de caracteres, es decir, en arrays de variables tipo char. Ahora mediante la función *gets()* se obtiene la cadena y a través de un bucle *for* se analiza cada carácter en busca de una de las cinco vocales. Por último, se imprime por pantalla la cantidad de veces que existe cada vocal en la cadena de texto introducida.

El bucle *for* se realizará hasta que se encuentre el carácter nulo (\0), o lo que es lo mismo, hasta que el carácter **exista**.

```

1 #include <stdio.h>
2 #include <ctype.h>
3 void main()
4 {
5     char cad[80];
6     int i, voc[5]={0};
7     puts("Proporcione una cadena de texto \
8 (80 caracteres max):");
9     gets(cad);
10
11    for(i=0; cad[i]; i++) //cad[i] => cad[i] != '\0'
12    {
13        switch(tolower(cad[i]))
14        {
15            case 'a':
16                voc[0]++;
17            case 'e':
18                voc[1]++;
19            case 'i':
20                voc[2]++;
21            case 'o':
22                voc[3]++;
23            case 'u':
24                voc[4]++;
25        }
26    }
27    printf("a=%d\n e=%d\n i=%d", voc[0], voc[1], voc[2]);
28    printf("\n o=%d\n u=%d", voc[3], voc[4]);
29 }

```

Código 43.- Problema 66

<https://i.ibb.co/GsGyX36/9-01.png>

<https://github.com/isidropo/EjerciciosC/blob/master/9.01.c>

Proporcioname una cadena de texto (80 caracteres max):
comprobacion de funcionamiento del programa
a=4
e=3
i=3
o=6
u=1

Salida 36.- Problema 66

<https://i.ibb.co/qBZnV7N/9-01E.png>

Aviso: La función `gets()` obtiene una cadena de texto incluyendo los espacios entre palabras hasta encontrar el carácter nueva línea (`\n`) e introduce el carácter nulo (`\0`) al final . Se ha de utilizar esta función en vez de `scanf()` que solo coge caracteres hasta encontrar el carácter nueva línea o espacio.

Nota: La función `tolower()` del archivo de cabecera `<ctype.h>` pasa cualquier carácter letra a minúscula. Resulta útil su uso para incluir tanto las vocales en mayúscula como en minúscula.

Nota: La función `puts()` imprime por pantalla el texto incluido entre comilla o guardado en una variable e introduce un carácter nueva línea (`\n`) al final, sin embargo, la función `printf()` solo imprime el texto sin cambiar el cursor de línea.

9.02. Problema 67. Realizar.

Realiza un programa que, mediante una cadena de texto introducida por teclado, se imprima por pantalla la misma cadena, pero con todos sus caracteres en mayúsculas. Mas tarde se quiere volver a imprimirla, pero sin espacios en blanco.

Nota: Resultaría muy tedioso comprobar cada carácter y hacer su conversión, pero gracias a la biblioteca de caracteres `<ctype.h>` se puede conseguir fácilmente. La función `toupper()` devuelve el carácter pasado, como parámetro, en mayúscula.

Aviso: La función `gets()` obtiene una cadena de texto incluyendo los espacios entre palabras hasta encontrar el carácter nueva línea (`\n`) e introduce el carácter nulo (`\0`) al final . Se ha de utilizar esta función en vez de `scanf()` que solo coge caracteres hasta encontrar el carácter nueva línea o espacio.

9.03. Problema 68. Explicar.

Solicitando una cadena de texto al usuario, se pide sumar **cada carácter numérico** y mostrar por pantalla la suma total.

Se me ocurren varias formas de solucionar este problema: **a)** restar a cada carácter numérico el carácter '0' y guardar el resultado en una variable entera, por ejemplo '2' – '0' (ASCII 50-48 respectivamente), **b)** realizar una función que acepte un carácter y mediante un selector `switch` devuelva para cada carácter numérico su valor entero correspondiente.



Se va a coger la opción b) simplemente por mostrar como una función puede actuar como un conversor, de hecho, la opción a) deja un código más sencillo.

Como solo se quieren sumar los caracteres numéricos será necesaria una función que nos diga si un carácter es o no numérico... por suerte dicha función está recogida en las bibliotecas de C y se denomina *isdigit()*.

Nota: La función *isdigit()* se encuentra en la biblioteca *<ctype.h>*. Se encarga de aceptar un carácter y comprobar si es o no numérico, es decir, devuelve 1 si lo es y 0 si no lo es. ¡Un consejo! la forma de utilización más frecuente de funciones que devuelven 1 o 0, es en sentencias de selección.

```

1 #include <stdio.h>
2 #include <ctype.h>
3 int carNum(char c)
4 {
5     switch(c)
6     {
7         case '0': return 0; case '1': return 1;
8         case '2': return 2; case '3': return 3;
9         case '4': return 4; case '5': return 5;
10        case '6': return 6; case '7': return 7;
11        case '8': return 8; case '9': return 9;
12    }
13 }
14 void main()
15 {
16     int suma=0;
17     char cad[80];
18     puts("\tSuma de caracteres numericos de \
19 una cadena");
20     gets(cad);
21
22     for(int i=0; cad[i]; i++)
23         if(isdigit(cad[i]))
24             suma += carNum(cad[i]);
25     printf("Suma: %d", suma);
26 }
```

Código 44.- Problema 68

<https://i.ibb.co/XyKg8fm/9-03.png>

<https://github.com/isidropo/EjerciciosC/blob/master/9.03.c>

```

Suma de caracteres numericos de una cadena
18352
Suma: 19
```

Salida 37.- Problema 68



<https://i.ibb.co/0cHsBzp/9-03E.png>

9.04. Problema 69. Desbloquear.

Se precisa pedir por teclado dos cadenas de texto. Una vez guardadas en variables tipo *char* de máximo 80 caracteres se quiere saber cuál de ella es mayor, desde el punto de vista del orden alfabético de los caracteres que las componen.

Nota: La función *strcmp()* incluida en la librería *<string.h>* compara las cadenas de texto s1 y s2 introducidas como parámetros. Si s1=s2 devuelve 0, si s1<s2 devuelve <0 y si s1>s2 devuelve >0.

La comparación de cadenas de hace carácter a carácter, entonces mediante el código ASCII, ¿Qué cadena es la mayor de las siguientes opciones?

Cadena 1:

aBCD V, ABCD

Cadena 2:

FFFF, H V

Cadena3:

aeiou, aeiuo V

Cadena 4:

x V, X

0/4 ¡0 aciertos! Parece que no se ha entendido el funcionamiento de la comparación de caracteres ASCII. La comparación se hace carácter a carácter y según el valor decimal recogido en la tabla ASCII un carácter será mayor, igual o menor que otro. Por ejemplo, para 'a' y 'e' cuyo valor decimal es 97 y 101 respectivamente... 'e' > 'a'.

1/4 2/4 Parece que no ha quedado claro del todo el funcionamiento de la comparación de caracteres ASCII. La comparación se hace carácter a carácter y según el valor decimal recogido en la tabla ASCII un carácter será mayor, igual o menor que otro. Por ejemplo, para 'a' y 'e' cuyo valor decimal es 97 y 101 respectivamente... 'e' > 'a'.

3/4 4/4 ¡Eso es! Comparando carácter a carácter de izquierda a derecha, según los valores de cada carácter en la tabla ASCII, se llega fácilmente la conclusión de que cadena es mayor.

```

1 #include <stdio.h>
2 #include <string.h>
3 void main()
4 {
5     char cad1[81], cad2[81];
6     puts("\tComparador de cadenas");
7     printf("Cadena 1: "); gets(cad1);
8     printf("Cadena 2: "); gets(cad2);
9
10    if(strcmp(cad1, cad2) == 0)
11        printf("\nCadena 1 == Cadena 2");
12    else if(strcmp(cad1, cad2) < 0)
13        printf("\nCadena 1 < Cadena 2");
14    else
15        printf("\nCadena 1 > Cadena 2");
16 }

```

Código 45.- Problema 69

<https://i.ibb.co/JyJCVhd/9-04.png>

<https://github.com/isidropo/EjerciciosC/blob/master/9.04.c>

```

Comparador de cadenas
Cadena 1: acde
Cadena 2: abcd

Cadena 1 > Cadena 2

```

Salida 38.- Problema 69

<https://i.ibb.co/R7mQ9tr/9-04E.png>

Final de los Problemas de C

¡Excelente! Has completado todos los _Problemas de C_. Si has cumplido con todas las tareas impartidas puedo afirmar que has adquirido un gran conocimiento sobre el lenguaje de programación de C.

Ahora te invito a ampliar tus conocimientos concretamente con problemas más extensos como los encontrados en una asignatura, examen, internet, etc.



___ ¡Un gran consejo! ___ Si el objetivo es aprobar una asignatura, consigue problemas de esta y realízalos, así te adaptarás a la metodología usada por el profesor y conseguirás una buena nota.