Tema 1

FUNDAMENTOS DE PROGRAMACIÓN

OBJETIVOS DOCENTES

Una vez estudiado el contenido del tema y realizados los ejercicios prácticos, debería saber:

- Discutir las ideas generales que han guiado la evolución de los lenguajes de programación.
- Discutir qué influencia ha tenido la arquitectura de la máquina de Von Neumann en los primeros lenguajes de programación imperativos.
- Discutir cómo se produce la ejecución de un programa en la máquina de Von Neumann.
- Discutir qué características tienen los lenguajes de bajo nivel (lenguaje máquina y ensamblador). Discutir las diferencias con los lenguajes de alto nivel y las ventajas de éstos frente a aquéllos.
- Discutir qué necesidades motivaron el desarrollo de los lenguajes de programación siguientes: FORTRAN, ALGOL, LISP, COBOL, BASIC, Visual BASIC, Prolog, SIMULA 67, Pascal, C, Modula-2, Ada, Smalltalk, C++, Java, sh, awk, Perl, JavaScript, PHP y Python. Discutir las principales características de estos lenguajes y qué nuevas capacidades aportó cada uno de ellos.
- Discutir en qué consiste la metodología de la programación estructurada.
- Discutir las características básicas de los cuatro paradigmas de programación fundamentales: programación imperativa, funcional, lógica y orientada a objetos.
- Discutir las características, y las ventajas y desventajas, de cada uno de los tres métodos siguientes de implementación de lenguajes de alto nivel: interpretación pura, compilación, y sistema híbrido de interpretación y compilación.
- Discutir en qué consiste el preprocesado del programa.

Tema 2

COMENZANDO A PROGRAMAR EN C++

OBJETIVOS DOCENTES

Una vez estudiado el contenido del tema y realizados los ejercicios prácticos, debería saber:

- Discutir el significado de cada una de las líneas de código de un programa sencillo, como es el caso del programa "Hola mundo!", que escribe en la consola una frase.
- Discutir cómo se construyen literales de tipo *string*. Discutir el uso de caracteres especiales en la construcción de literales de tipo *string*.
- Discutir cómo se ponen datos en el flujo estándar de salida, empleando manipuladores.
- Discutir por qué los flujos de entrada y salida funcionan como buffers.
- Discutir la finalidad de los flujos de salida, entrada y error estándar.

Tema 3

VARIABLES Y TIPOS DE DATOS: PRINCIPIOS BÁSICOS

OBJETIVOS DOCENTES

Una vez estudiado el contenido del tema y realizados los ejercicios prácticos, debería saber:

- Discutir qué es una variable y qué atributos tiene. En particular, discutir el significado de los atributos nombre, dirección en memoria, valor y tipo de dato.
- Discutir qué es la declaración e inicialización de una variable.
- Discutir qué son las variables constantes. Discutir qué significa que el lenguaje permita establecer ligaduras estáticas y dinámicas sobre el valor de las constantes.
- Discutir qué es un bloque de código, y los conceptos de ámbito y visibilidad de una variable.
- Discutir la diferencia entre variables locales y globales.
- Discutir qué motivaciones tiene la declaración de los tipos de datos de las variables.
- Discutir qué son los tipos de datos primitivos. En particular, qué son los tipos número entero, número real, Booleano y carácter.
- Discutir las facilidades que los diferentes lenguajes proporcionan al programador para que éste defina sus propios tipos de datos.
- Discutir qué es un array, qué significa declarar e inicializar un array, cómo se accede en el programa a los componentes de un array y qué maneras tienen las implementaciones de almacenar los arrays bidimensionales en memoria.
- Discutir la manera en que diferentes lenguajes soportan las cadenas de caracteres, bien almacenándolas en arrays, cuyos componentes son de tipo carácter, o bien mediante objetos del tipo de dato *string*.
- Discutir qué son los punteros y cuáles son algunos de sus principales usos.
- Discutir qué son las variables en memoria dinámica, y cuál es su diferencia con las variables locales y globales.
- Discutir algunos de los errores típicos de programación que se cometen al trabajar con arrays, punteros y variables en memoria dinámica. Explicar las facilidades que proporcionan diferentes lenguajes para la prevención y diagnosis de este tipo de errores, que en ocasiones son cometidos por el programador.

Tema 4

VARIABLES Y TIPOS DE DATOS: PROGRAMACIÓN EN C++

OBJETIVOS DOCENTES

Una vez estudiado el contenido del tema y realizados los ejercicios prácticos, debería saber:

- Discutir cómo y en qué partes del programa se realiza la declaración e inicialización de variables en C++.
- Discutir qué restricciones impone C++ a los nombres de los identificadores.
- Discutir cómo se realiza la declaración a las variables constantes y cómo se les asigna valor.
- Discutir qué tipos de datos básicos (también llamados primitivos) proporciona C++.
- Realizar programas sencillos, en los cuales se declaren e inicialicen variables de los tipos de datos básicos, y se vuelque el valor de dichas variables a la consola.
- Discutir qué son los límites numéricos de los tipos de datos básicos y realizar programas para obtenerlos.
- Discutir cómo la implementación de C++ realiza la inicialización por defecto de las variables locales y globales de los tipos de datos básicos.
- Discutir cómo se declaran tipos enumerados y declarar variables de este tipo en programas.
- Discutir qué son las estructuras, y realizar la declaración e inicialización de estructuras en programas.
- Declarar e inicializar arrays en programas. Acceder a los componentes de un array.
- Declarar e inicializar variables y constantes de los tipos de datos `std::string` y `std::vector`, que están declarados en la librería estándar de C++.
- Discutir las diferencias entre arrays y vectores.
- Discutir de qué tipo son los flujos predefinidos de entrada y salida.
- Discutir cómo se declaran los punteros y cómo se emplean para referenciar variables, incluyendo las variables en memoria dinámica.

- Discutir cómo se declaran variables y arrays en memoria dinámica, usando el operador **new**, y cómo se libera el espacio ocupado por esas variables usando el operador **delete**. Reservar y liberar espacio para este tipo de variables en los programas.
- Discutir cómo se define en C++ el ámbito y la visibilidad de las variables. Discutir el acceso a variables globales mediante el operador ámbito. Aplicar estos conceptos en el desarrollo de programas.

Tema 5

ASIGNACIONES Y EXPRESIONES: PRINCIPIOS BÁSICOS

OBJETIVOS DOCENTES

Una vez estudiado el contenido del tema y realizados los ejercicios prácticos, debería saber:

- Discutir qué es una sentencia de asignación.
- Discutir qué es una expresión. Clasificar las expresiones en aritméticas, relacionales, Booleanas y condicionales.
- Discutir qué son los operadores relacionales y los Booleanos. Indicar cuáles son.
- Discutir la diferencia de significado entre el operador de asignación y el operador de comparación de igualdad.
- Clasificar los operadores, en función del número de sus operandos, en unarios, binarios y ternarios.
- Clasificar la notación de las expresiones, en función de la posición relativa de los operadores respecto a los operandos, en notación prefija, sufija e infija.
- Emplear en asignaciones los operadores que combinan una operación aritmética con la asignación (`+=`, `-=`, `*=`, `/=`, `%=`).
- Emplear en expresiones los operadores incremento `++` y decremento `--`, usados tanto como prefijo como postfijo.
- Discutir qué finalidad tienen las reglas de asociatividad y precedencia. Discutir las reglas de precedencia que comúnmente se aplican en los lenguajes de programación.
- Discutir qué es el sistema de tipos de un lenguaje de programación.
- Discutir qué es la sobrecarga de los operadores.
- Discutir la diferencia entre las conversiones de tipo explícitas e implícitas. Discutir las reglas que aplica el compilador de C para realizar las coerciones (conversiones implícitas).
- Discutir en qué consiste la verificación de tipos, y la diferencia entre verificación estática y dinámica.
- Discutir qué se entiende por lenguaje fuertemente tipado.

Tema 6

ASIGNACIONES Y EXPRESIONES: PROGRAMACIÓN EN C++

OBJETIVOS DOCENTES

Una vez estudiado el contenido del tema y realizados los ejercicios prácticos, debería saber:

- Discutir cómo son las sentencias de asignación en C++ y las conversiones de tipo que se producen automáticamente al ejecutar dichas sentencias.
- Construir expresiones empleando los operadores aritméticos de C++. Decidir, aplicando las reglas de precedencia, el orden de evaluación de los operadores en las expresiones aritméticas.
- Construir expresiones lógicas empleando los operadores aritméticos, relacionales y lógicos de C++. Decidir, aplicando las reglas de precedencia, el orden de evaluación de los operadores que intervienen en las expresiones lógicas. Discutir en qué consiste la evaluación en cortocircuito de los operadores lógicos.
- Discutir el significado de los operadores `>>` y `<<` cuando se aplican a palabras de bits y a flujos.
- Emplear las funciones matemáticas declaradas en la cabecera estándar *cmath*.
- Emplear punteros para el direccionamiento de variables. Emplear los operadores dirección-de e indirección. Discutir la relación existente entre punteros y arrays. Emplear punteros para direccionar los componentes de un array.
- Declarar variables del tipo `std::complex`, `std::string` y `std::vector`, inicializarlas y usar dichas variables en la construcción de expresiones, empleando las funciones miembro y los operadores definidos en la librería estándar para este tipo de datos.
- Discutir qué son los iteradores. Declarar iteradores a vectores y emplearlos para direccionar los componentes de un vector.
- Emplear variables de tipo estructura en expresiones y sentencias de asignación. Emplear punteros para referenciar las estructuras y sus miembros. Emplear arrays cuyos componentes sean de tipo estructura. Discutir qué es una estructura autorreferenciada y cuál es su utilidad.
- Realizar programas sencillos en los que se realicen operaciones de entrada por teclado y salida por consola, se declaren e inicialicen variables, se manipulen los datos almacenados en variables empleando expresiones y se guarden en variables los resultados, usando para ello sentencias de asignación.

Tema 7

CONTROL DEL FLUJO DEL PROGRAMA: PRINCIPIOS BÁSICOS

OBJETIVOS DOCENTES

Una vez estudiado el contenido del tema y realizados los ejercicios prácticos, debería saber:

- Discutir qué finalidad tienen las sentencias de selección y las sentencias iterativas, y cuál es la diferencia entre ambas.
- Discutir cómo se controla el flujo del programa en las sentencias de selección **if**, **case** y **switch**. Discutir las diferencias entre ellas.
- Discutir en qué consiste el problema del **else** ambiguo y de qué maneras evitan los lenguajes que surja este problema.
- Discutir el significado y qué usos tienen las sentencias **break** y **continue**.
- Discutir cómo se controla el flujo del programa en las diferentes formas de la sentencia iterativa **for**, y en las sentencias controladas mediante expresión Booleana. Respecto a este último tipo de sentencias, discutir la diferencia entre las sentencias con precondición (por ejemplo, sentencia **while**) y con postcondición (por ejemplo, sentencia **do-while**).
- Discutir qué son las excepciones y en qué consiste, a grandes rasgos, la captura y tratamiento de las excepciones.

Tema 8

CONTROL DEL FLUJO DEL PROGRAMA: PROGRAMACIÓN EN C++

OBJETIVOS DOCENTES

Una vez estudiado el contenido del tema y realizados los ejercicios prácticos, debería saber:

- Emplear las sentencias de selección (**if** y **switch**) e iterativas (**for**, **while** y **do-while**), así como las sentencias **break** y **continue**, en la escritura de programas en C++.
- Realizar programas en los que se capturen y traten excepciones dentro del código de la función **main**. Las excepciones capturadas podrán ser tanto las lanzadas por el programador, como las lanzadas por las funciones y operadores de la librería estándar de C++.
- Declarar variables en memoria dinámica, capturando y tratando la excepción `std::bad_alloc`.
- Programar la entrada de datos a través del flujo `std::cin`, empleando un bucle **while**, analizando el tipo de error producido y descartando el contenido del flujo de entrada cuando proceda. Discutir cómo cambia el estado del flujo de entrada, dependiendo de si la operación sobre el flujo se ha realizado o no con éxito. Emplear en los programas las funciones que proporcionan información acerca del estado del flujo de entrada y que cambian dicho estado.
- Escribir programas en C++ en los cuales se realice entrada y salida a fichero de texto.

Tema 9

SUBPROGRAMAS: PRINCIPIOS BÁSICOS

OBJETIVOS DOCENTES

Una vez estudiado el contenido del tema y realizados los ejercicios prácticos, debería saber:

- Discutir qué es un subprograma y qué ventajas proporciona el uso de subprogramas.
- Discutir de qué partes consta la definición de una función.
- Discutir cómo se realiza la invocación de una función. Definir qué son los parámetros formales de la función y qué son los parámetros actuales usados en la invocación.
- Discutir cómo se realiza la evaluación de una función. Discutir en qué consiste el paso de parámetros por valor y por referencia, y la diferencia entre ambas formas de invocación de funciones.
- Discutir cómo se realiza el paso de parámetros a funciones en los lenguajes C y C++.
- Reconocer el ámbito y la visibilidad de las variables locales y de los parámetros declarados en funciones.
- Reconocer si una función es recursiva, si tiene recursividad lineal y si tiene recursividad de cola.
- Discutir la diferencia entre una función y un procedimiento.
- Discutir de qué partes consta la definición de un procedimiento y cómo se realiza su invocación.

Tema 10

SUBPROGRAMAS: PROGRAMACIÓN EN C++

OBJETIVOS DOCENTES

Una vez estudiado el contenido del tema y realizados los ejercicios prácticos, debería saber:

- Discutir cómo se definen e invocan funciones en lenguaje C++. Discutir cómo se realiza el paso de parámetros a las funciones en lenguaje C++.
- Realizar programas en los cuales se definan e invoquen funciones.
- Dada una función, reconocer el ámbito y la visibilidad de los parámetros formales y las variables locales.
- Discutir cómo son tratadas, a efectos de su almacenamiento en memoria, las variables estáticas declaradas en el cuerpo de una función. Discutir cuál es el ámbito de dichas variables. Emplear este tipo de variables en la programación de funciones.
- Discutir el significado de la sentencia **return** y emplear dicha sentencia en la programación de funciones.
- Realizar programas en los cuales se definan funciones que lancen excepciones y en los cuales estas excepciones sean capturadas y tratadas.
- Discutir qué es la declaración y la definición de una función, y cuál es la diferencia entre ambas. Realizar programas en los cuales las funciones sean declaradas y definidas.
- Escribir programas organizados en varios ficheros.
- Discutir cuál es el propósito de los espacios de nombres, y realizar programas en los cuales se definan espacios de nombres y se usen las entidades declaradas en ellos.

Tema 11

ESTRUCTURAS DE DATOS: PRINCIPIOS BÁSICOS

OBJETIVOS DOCENTES

Una vez estudiado el contenido del tema y realizados los ejercicios prácticos, debería saber:

- Discutir qué son los tipos abstractos de datos y qué utilidad tienen.
- Para las estructuras de datos *lista*, *pila*, *cola*, *mapa* y *árbol*, discutir:
  - Qué características tiene cada estructura. Los conceptos y la terminología básicos relacionados con cada estructura de datos.
  - Qué utilidad puede tener cada una de estas estructuras de datos en la realización de programas.
  - Qué operaciones se realizan comúnmente sobre cada una de estas estructuras de datos.
  - Cómo puede implementarse cada una de estas estructuras de datos mediante un array y mediante estructuras autorreferenciadas. Pros y contras en cada caso de estas dos opciones de implementación.
- Programar en C++ listas, incluyendo pilas y colas, mediante estructuras autorreferenciadas, así como programar funciones que realicen operaciones básicas sobre listas.

Tema 12

ESTRUCTURAS DE DATOS: PROGRAMACIÓN EN C++

OBJETIVOS DOCENTES

Una vez estudiado el contenido del tema y realizados los ejercicios prácticos, debería saber:

- Planteado un problema de programación, discutir qué estructuras de datos sería más adecuado usar de entre las siguientes: listas, pilas, colas y mapas.
- Discutir conceptos básicos de los componentes fundamentales de la Standard Template Library (STL) de C++. Esto es, de los contenedores, algoritmos e iteradores.
- Emplear en la realización de programas en C++ los tipos lista, cola, pila y mapa declarados en la STL, sabiendo declarar e inicializar variables de esos tipos, insertar, eliminar, modificar y buscar elementos, y manipular el contenido de las variables de dichas estructuras de datos empleando funciones miembro.
- Emplear iteradores para acceder a los elementos de listas, colas, pilas y mapas.

Tema 13

ALGORITMOS: PRINCIPIOS BÁSICOS

OBJETIVOS DOCENTES

Una vez estudiado el contenido del tema y realizados los ejercicios prácticos, debería saber:

- Discutir qué es un algoritmo.
- Discutir las características básicas de los siguientes paradigmas para el diseño de algoritmos: fuerza bruta o búsqueda exhaustiva, divide y vencerás, programación dinámica, programación lineal, programación entera, y búsqueda y enumeración.
- Discutir las principales características de las dos formas siguientes de describir los algoritmos: mediante pseudocódigo y mediante diagrama de flujo.
- Discutir cómo se estima la complejidad de un algoritmo y qué es la notación **O**.
- Discutir cuál es la finalidad de los algoritmos de ordenación.
- Programar en C++ los tres algoritmos de ordenación siguientes: método de la burbuja, ordenación por inserción y ordenación por mezcla.

Tema 14

ALGORITMOS: PROGRAMACIÓN EN C++

OBJETIVOS DOCENTES

Una vez estudiado el contenido del tema y realizados los ejercicios prácticos, debería saber:

- Realizar programas en C++ empleando los algoritmos indicados a continuación, que se encuentran declarados en la STL de C++.

  `std::count`

  `std::count_if`

  `std::remove_copy`

  `std::replace_copy`

  `std::reverse`

  `std::transform`

  `std::sort`

  `std::find_if`

- Emplear expresiones lambda para definir funciones anónimas y pasarlas como argumento a algoritmos.