Jornada Matutina

liceo compu market  villa nueva

jonathan amadeo blanco cano

5to bachillerato

**HISTORIA DE LA COMPUTACIÓN**

***Antecedentes***

El concepto numérico se considera, generalmente, como anterior al desarrollo de los lenguajes

escritos; los primeros registros del hombre son anotaciones sobre la cantidad de granos,

animales y demás posesiones personales. Con este fin, el hombre empleaba guijarros, palos y/o

marcas en las paredes de las cavernas que habitaba.

El deseo humano de obtener mayor información y mejores comunicaciones, gradualmente fue

dejando atrás estos sencillos instrumentos. El ábaco es considerado como el primer instrumento

elaborado por el hombre para realizar operaciones aritméticas de manera más eficiente. El ábaco

no realiza cálculos de manera autónoma (por si solo), simplemente le permite al hombre

realizarlos de manera más eficiente.

La primera máquina que realizaba cálculos de manera autónoma fue inventada en 1642 por el

gran matemático y filósofo francés Blas Pascal. La máquina de Pascal (en honor a su inventor),

era movida mediante una serie de ruedas dentadas, numeradas del cero al nueve, alrededor de

sus circunferencias y era capaz de sumar y restar en forma directa, mostrando un número a

través de una ventanita y por este hecho tiene la ventaja de evitar tener que contar, como en el

caso del ábaco; además, presenta los resultados en forma más accesible.

**Figura 1.**

En 1671 Gottfried Wilhelm Leibniz le adicionó a la máquina de Pascal un cilindro diseñado

especialmente para que fuera capaz de multiplicar y dividir de manera directa.

A principios del siglo XIX el ingeniero Joseph Marie Jacquard perfeccionó el concepto de tarjeta

perforada, con el cual se podían “programar” las máquinas de tejer para que siguieran un patrón

o diseño. Este concepto fue posteriormente utilizado en las máquinas de cómputo para decirles

qué debían calcular.

**¿Qué es la programación?**

|  |
| --- |
| Se conoce como programación de computadores a la implementación de un algoritmo en un determinado lenguaje de programación, conformando un programa. Mientras que un algoritmo se ejecuta en una máquina abstracta que no tiene limitaciones de memoria o tiempo, un programa se ejecuta en una máquina real, que sí tiene esas limitaciones. El lenguaje de programación puede ser de alto nivel, medio nivel o bajo nivel, en función del grado de abstracción.  Pero ninguno de estos avances fue antecesor directo de las computadoras electrónicas de hoy.  La verdadera precursora de la computadora fue la máquina llamada “motor de diferencias”,  construida en 1822 por Charles Babbage para calcular algoritmos y tablas astronómicas.  A partir de su trabajo en el motor de diferencias, Babbage diseñó un poderoso instrumento para  el cálculo automático. Tal como lo concibió Babbage, este “motor analítico” estaría impulsado  por vapor, y trabajaría basado en un programa de planeación almacenado en tarjetas  perforadas. Ésta máquina estaba dividida funcionalmente en dos grandes partes: una que  ordenaba y otra que ejecutaba las órdenes. La que ejecutaba las órdenes era una versión muy  ampliada de la máquina de Pascal, mientras que la otra era la parte clave. La innovación  consistía, en que el usuario podía combinando las especificaciones de control, lograr que la  misma máquina ejecutara operaciones complejas, diferentes a las hechas antes. Babbage  concibió una memoria, un procesador aritmético, los medios de ingresar datos y/o instrucciones,  así como una sección de producción que imprimiría los resultados.  Todos estos son los elementos de las computadoras modernas y no se hicieron realidad sino  varias generaciones después de ser propuestos por Babagge. Charles Babbage no pudo  implementar el motor analítico y murió sin saber que realmente funcionaba.  **Figura 2. Máquina Diferencial de Charles Babbage**  Durante los cien años siguientes, las máquinas activadas por tarjetas perforadas se modificaron,  se mejoraron e hicieron más rápidas, pero aún no podían mantener el ritmo de las crecientes  necesidades humanas de procesamiento de listas de pagos, cuentas, facturas, análisis de  ventas y otros problemas.  En 1937, Howard H. Aiken, un candidato al doctorado en física de Harvard, trabajó en una  máquina que podría resolver automáticamente ecuaciones diferenciales. La *International*  *Business Machines* (I.B.M.), hoy una de las más grandes empresas de esta fase de la industria  norteamericana, ayudó al inventor a crear la “Calculadora Controlada de Secuencia Automática”  conocida como el “Mark I”.  El Mark I era un monstruo de cuatro y media toneladas métricas, con 78 aparatos  independientes vinculados por unos 800 kilómetros de cable. En tres décimas de segundo podía  efectuar sumas y restas de 23 dígitos y en cerca de 6 segundos podía multiplicar números de 23  dígitos. Fue retirado en 1959.  Tanto el Mark I como la Segunda Guerra Mundial, desempeñaron un papel clave en el desarrollo  de las computadoras. El Mark I aportó los ingredientes tecnológicos básicos mientras que la  segunda guerra mundial con sus inmensas demandas de mano de obra y máquinas, creó la  necesidad. El resultado fue el “Integrador y Calculador Numérico Electrónico”, más conocido  como el “ENIAC”.  **Figura 4. ENIAC**  Terminado en 1946, el ENIAC fue creado para el ejército norteamericano en la escuela Moore de  Ingeniería Eléctrica, de la Universidad de Pennsylvania. Sus creadores fueron un estudiante  graduado, J. Presper Eckert, y un físico, el Dr. John W. Mauchly. Juntos eliminaron la necesidad  de las partes que se movían mecánicamente en la computadora central. En su lugar, adaptaron  circuitos eléctricos de gatillo “flip-flop” y “pulsaciones” electrónicas para conectar o desconectar  tubos al vacío, como interruptores.  Como las interrupciones de este tipo podían hacerse miles de veces más rápido que los aparatos  electro-mecánicos, el ENIAC constituyó un gran inicio hacia el desarrollo de las computadoras  modernas.  El último paso para completar el concepto de la computadora de hoy, fue el desarrollo del  concepto de máquina almacenadora de programas. Este paso se dio a fines de los cuarenta,  después de que el célebre matemático hungaro-norteamericano Dr. John Von Neumann sugirió  que las instrucciones de operación, así como los datos, se almacenaran de la misma manera en  la “memoria” de la computadora. Además, aportó la idea de hacer que la computadora  modificara sus propias instrucciones de acuerdo con un control programado. Las ideas de Von  Newman fueron fundamentales para los desarrollos posteriores y se le considera el padre de las  computadoras. Desde entonces, se ha tratado de modificar, mejorar y apresurar estos  conceptos, en fin, de hacer computadoras cada vez más eficientes.  4  **Figura 5. Computador electrónico – IBM 360**  Existen básicamente dos tipos básicos de computadoras: las análogas y las digitales. También  existen sistemas llamados híbridos que emplean elementos tanto análogos como digitales.  Los fenómenos que se comportan en forma continua reciben el nombre de analógico por  ejemplo: la altura de una columna de mercurio en un termómetro clínico, puede variar entre las  marcas de treinta y cuarenta y cinco grados y en todo momento puede estar en cualquier punto  intermedio de la escala, lo mismo ocurre con un voltaje eléctrico o la rotación angular de un eje.  En una computadora análoga los números están representados por cantidades físicas  continuamente variables como las anteriores. Tales máquinas tienen aplicaciones físicas e  industriales que representan procesos físicos que ocurren con el paso del tiempo.  Existe otro tipo de fenómenos ejemplo: si se averigua la cantidad de ventanas de un edificio se  llegará a la conclusión de que son un número exacto como 90 y que no puede haber 90 y  media. Estos fenómenos reciben el nombre de digitales porque dan la idea de que se pueden  cuantificar con los dedos de la mano. La computadora digital opera con números representados  directamente en forma “digital”. Tales computadoras son las más extensamente usadas y  pueden aplicarse en todos los campos que requieren operaciones aritméticas y manejo de  información  **ESTRUCTURA DE UN COMPUTADOR**  Un **COMPUTADOR** es una máquina que realiza cálculos de manera automática. Se divide  fundamentalmente en dos partes: el hardware y el software. El **HARDWARE** es la parte física de  un computador, es decir, la parte que realiza los cálculos. El **SOFTWARE** es la parte lógica del  computador, es decir, la parte que le dice al hardware qué hacer. Usando una metáfora se puede  decir que: *“Un computador es como un ser humano: el hardware es el cuerpo y el software es*  *la mente”*  **1.3.1. *Arquitectura de hardware***  Un computador desde la perspectiva del hardware, está constituido por una serie de dispositivos  cada uno con un conjunto de tareas definidas. Los dispositivos de un computador se dividen  según la tarea que realizan en: dispositivos de entrada, dispositivos de salida, dispositivos de  comunicaciones, dispositivos de almacenamiento y dispositivo de cómputo.  7  **Figura 11. Arquitectura de Hardware**  **Dispositivos de entrada:** Son aquellos que permiten el ingreso de datos a un computador.  Entre estos se cuentan: teclados, ratones, scaners, micrófonos, cámaras fotográficas, cámaras  de video, controles de juegos, lápices ópticos, y guantes de realidad virtual.  **Figura 12. Dispositivos de Entrada**  **Dispositivos de salida.** Son aquellos que permiten mostrar información almacenada o  procesada por el computador. Entre otros están: las pantallas de video, impresoras, audífonos,  plotters, guantes de realidad virtual, gafas y cascos virtuales.  **Figura 13. Dispositivos de Salida**  **Dispositivos de almacenamiento.** Son aquellos en los cuales el computador puede guardar  información y de los cuales puede obtener información previamente almacenada. Entre otros  están los discos flexibles, discos duros, unidades de cinta, CD-ROM, CD-ROM de re-escritura y  DVD.  8  **Figura 14. Dispositivos de Almacenamiento.**  **Dispositivos de comunicación:** Son aquellos que le permiten a un computador comunicarse  con otros. Entre estos se cuentan los modems y tarjetas de red.  **Figura 15. Modem**  **Dispositivo de cómputo:** Es la parte del computador que realiza todos los cálculos y tiene el  control sobre los demás dispositivos. Está formado por tres elementos fundamentales: la unidad  central de proceso, la memoria y el bus de datos y direcciones.  **Figura 16. Diagrama esquemático del dispositivo de computo**  La **unidad central de proceso (UCP)1:** es el ‘cerebro’ del computador, está encargada de  realizar todos los cálculos, utilizando para ello la información almacenada en la memoria y de  controlar los demás dispositivos, procesando las entradas y salidas provenientes y/o enviadas a  los mismos. Mediante el bus de datos y direcciones, la UCP se comunica con los diferentes  dispositivos enviando y obteniendo tales entradas y salidas.  Para realizar su tarea la unidad central de proceso dispone de una unidad aritmético lógica, una  unidad de control, un grupo de registros y opcionalmente una memoria caché para datos y  direcciones.  La ***unidad aritmético lógica (UAL)***2 es la encargada de realizar las operaciones aritméticas y  lógicas requeridas por el programa en ejecución, la ***unidad de control*** es la encargada de  determinar las operaciones e instrucciones que se deben realizar, el ***grupo de registros*** es  donde se almacenan tanto datos como direcciones necesarias para realizar las operaciones  requeridas por el programa en ejecución y la *memoria caché* se encarga de mantener  direcciones y datos intensamente usados por el programa en ejecución.  1 La unidad central de proceso es más conocida como CPU por sus siglas en inglés Central Process Unit.  2 La unidad aritmético lógica es más conocida como ALU por sus siglas en inglés Arithmetic Logic Unit.  9  **Figura 17. Unidad Central de Proceso.**  La **memoria** está encargada de almacenar toda la información que el computador está usando,  es decir, la información que es accedida (almacenada y/o recuperada) por la UCP y por los  dispositivos.  La unidad de medida de memoria es el byte, constituido por 8 bits (ceros o unos). Cada byte  tiene asignada una dirección de memoria, para poder ser accedida por la UCP. Para la  interpretación de la información que está en memoria, como datos o comandos o instrucciones,  se utilizan códigos que la UCP interpreta para llevar a cabo las acciones deseadas por el  usuario.  Existen diferentes tipos de memoria, entre las cuales se encuentran las siguientes:  •• **RAM** (Random Access Memory): Memoria de escritura y lectura, es la memoria principal del  computador. El contenido solo se mantiene mientras el computador está encendido.  •• **ROM** (Read Only Memory): Memoria de solo lectura, es permanente y no se afecta por el  encendido o apagado del computador. Generalmente almacena las instrucciones que le  permiten al computador iniciarse y cargar (poner en memoria RAM) el sistema operativo.  •• **Caché**: Memoria de acceso muy rápido, usada como puente entre la UCP y la memoria RAM,  para evitar las demoras en la consulta de la memoria RAM.  El **bus de datos y direcciones** permite la comunicación entre los elementos del computador.  Por el bus de datos viajan tanto las instrucciones como los datos de un programa y por el bus de  direcciones viajan tanto las direcciones de las posiciones de memoria donde están instrucciones  y datos, como las direcciones lógicas asignadas a los dispositivos.  **1.3.2. *Arquitectura de software***  Un computador desde la perspectiva del software, está constituido por:  • Un sistema operativo.  • Un conjunto de lenguajes a diferente nivel con los cuales se comunica con el usuario y con  sus dispositivos. Entre estos están los lenguajes de máquina, los ensambladores y los de  alto nivel.  • Un conjunto de aplicaciones de software.  • Un conjunto de herramientas de software.  10  **Software:** Es un conjunto de instrucciones que le dicen al hardware que hacer. El hardware por  si solo no puede hacer nada.  **Lenguaje de programación:** Es un conjunto de reglas y estándares que es utilizado para  escribir programas de computador (software), que puedan ser entendidos por él.  **Programa:** Es la representación de algún software en un lenguaje de programación específico.  *1.3.2.1.* ***Sistema Operativo***  Es el software encargado de administrar los recursos del sistema. Adicionalmente, ofrece un  conjunto de comandos para interactuar con la máquina.  Los sistemas operativos pueden ser escritos en lenguaje de alto nivel (UNIX fue escrito en C), en  lenguaje ensamblador y/o en lenguaje máquina. Algunos de los sistemas operativos más  conocidos son DOS, UNIX, LINUX y las distintas versiones de Microsoft Windows.  *1.3.2.2.* ***Lenguajes a diferente nivel***  ***1.3.2.2.1 Lenguaje de Máquina***  Es el único lenguaje que entiende el hardware (máquina) y usa exclusivamente el sistema  binario (ceros y unos). Este lenguaje es específico para cada hardware (procesador, dispositivos,  etc.).  El programa (tanto códigos de instrucción como datos) es almacenado en memoria. La  estructura de una instrucción en lenguaje máquina es la siguiente:  **CODIGO ARGUMENTO(S)**  **Figura 18. Lenguaje de máquina**  ***1.3.2.2.2 Lenguaje Ensamblador***  Es un lenguaje que usa mnemónicos (palabras cortas escritas con caracteres alfanuméricos),  para codificar las operaciones. Los datos y/o direcciones son codificados generalmente como  números en un sistema hexadecimal. Generalmente es específico (aunque no único) para cada  lenguaje de máquina. La estructura de una instrucción en este lenguaje es la siguiente:  **MNEMONICO ARGUMENTO(S)**  Un **ENSAMBLADOR** es un software, generalmente escrito en lenguaje de máquina, que es  capaz de traducir de lenguaje ensamblador a lenguaje de máquina.  11  **Figura 19. Lenguaje Ensamblador.**  ***1.3.2.2.3 Lenguaje de Alto Nivel***  Es un lenguaje basado en una estructura gramatical para codificar estructuras de control y/o  instrucciones. Cuenta con un conjunto de palabras reservadas (escritas en lenguaje natural).  Estos lenguajes permiten el uso de símbolos aritméticos y relacionales para describir cálculos  matemáticos, y generalmente representan las cantidades numéricas mediante sistema decimal.  Gracias a su estructura gramatical, estos lenguajes permiten al programador olvidar el  direccionamiento de memoria (donde cargar datos y/o instrucciones en la memoria), ya que este  se realiza mediante el uso de conceptos como el de variable.  Los **COMPILADORES** e I**NTERPRETES** son software capaz de traducir de un lenguaje de alto  nivel al lenguaje ensamblador específico de una máquina. Los primeros toman todo el programa  en lenguaje de alto nivel, lo pasan a lenguaje ensamblador y luego lo ejecutan. Los últimos  toman instrucción por instrucción, la traducen y la van ejecutando.  **Figura 20. Lenguaje de Alto Nivel.**  *1.3.2.3.* ***Aplicaciones***  Una **APLICACION** es un software construido para que el computador realice una tarea  específica y con el cual no se puede construir otro software. Ejemplos de aplicaciones son los  procesadores de texto como *Microsoft Word* y *Word Perfect* y las hojas electrónicas de cálculo  como *Microsoft Excel* y *Lotus*.  *1.3.2.4.* ***Herramientas***  Una **HERRAMIENTA** es un software construido especialmente para el desarrollo de nuevo  software, (tanto de aplicaciones como de herramientas). Ejemplos de herramientas son los  compiladores como Turbo C, Turbo Pascal y Dev C++, las herramientas CASE y los ambientes  integrados de desarrollo.  12  **Figura 21. Arquitectura de Software Completa.**  **1.4. RESUMEN.**  Las computadoras son un avance de los sencillos utensilios que el nombre uso para contar a  principios de su historia.  El ábaco fué el primer calculador digital.  La máquina de Pascal inventada en 1642 fue la primera máquina calculadora.  La verdadera precursora de las computadoras electrónicas fue el “Motor de Diferencias”  construida en 1922 por Charles Babbage.  En 1937 la IBM ayudó a Howard Aiken a crear el Mark I.  El Mark I y la Segunda Guerra Mundial desempeñaron papeles claves en el desarrollo de las  computadoras norteamericanas.  El resultado fue el Eniac terminado en 1946 creado por un estudiante graduado Prespert Eckert  y el físico John Mauchly.  Afines de los años 40 John Von Nuemann concibió la idea de que en la memoria coexistan datos  con instrucciones. Alrededor de este concepto y el de control programado gira toda la evolución  posterior de la industria de las computadoras.  Existen computadoras analógicas y digitales. Son computadoras digitales aquellas que manejan  la información de manera discreta y son analógicas las que trabajan por medio de funciones  continuas, generalmente representación de señales eléctricas.  El computador está constituido por Hardware y Software. El hardware es la parte física y el  software la parte lógica.  Los componentes del hardware son: dispositivos de entrada, dispositivos de salida, dispositivos  de almacenamiento, dispositivos de comunicación y dispositivo de cómputo (Unidad central de  proceso, memoria, bus de datos y direcciones).  Desde la perspectiva de software el computador está constituido por: sistema operativo,  conjunto de lenguajes a diferente nivel (lenguajes de máquina, ensambladores y alto nivel),  aplicaciones y herramientas.  **1.5. BIBLIOGRAFÍA.**  • BECERRA C., *Algoritmos: Conceptos Básicos*, 4ª edición, 1998.  **1.6. LECTURAS COMPLEMENTARIAS.**  **NTRODUCCIÓN A LOS ALGORITMOS**  **2.1. OBJETIVOS**  Presentar una definición informal del concepto de algoritmo que será adoptada para el resto del  texto. Igualmente, clarificar los tipos de problemas que pueden ser resueltos de manera  algorítmica, y ejemplificar el trabajo que implica resolver un problema mediante un programa de  computador. Los ejercicios del final del capítulo ofrecen una oportunidad para que el estudiante  consolide estos conceptos y se ejercite en relacionar la información dada en el enunciado de un  problema, con la información desconocida.  También, se presentan los elementos básicos de los dos principales formalismos que serán  utilizados para especificar algoritmos que se quieren ejecutar en un computador: el  pseudocódigo y los diagramas de flujo.  **2.2. INTRODUCCIÓN**  Diariamente el ser humano trata de darle solución a cada problema que se le presenta, o de  mejorar las soluciones disponibles. Para algunos problemas fundamentales ha encontrado  soluciones brillantes que consisten en una serie de acciones, que siempre que se realicen de  manera ordenada y precisa conducen a la respuesta correcta. Algunas de esas soluciones han  requerido el trabajo, la inteligencia y la persistencia de muchas generaciones. Hoy la sociedad  cuenta con ese legado de soluciones, además de las máquinas capaces de ejecutarlas precisa y  velozmente. Esas máquinas maravillosas, de las cuales trata el capítulo 1, son los  computadores.  **2.3. ALGORITMOS**  La palabra algoritmo se deriva de **Al-khôwarizmi**, un matemático y astrónomo del siglo IX  quien al escribir un tratado sobre manipulación de números y ecuaciones, el ***Kitab al-jabr***  ***w’almugabala***, usó en gran medida la noción de lo que se conoce hoy como algoritmo.  Un ALGORITMO es una secuencia finita ‘bien definida’ de tareas ‘bien definidas’, cada una de las  cuales se puede realizar con una cantidad finita de recursos. Se dice que una tarea esta ‘bien  definida’, si se saben de manera precisa las acciones requeridas para su realización. Aunque los  recursos que debe utilizar cada tarea deben ser finitos estos no están limitados, es decir, si una  tarea bien definida requiere una cantidad inmensa (pero finita) de algún recurso para su  realización, dicha tarea puede formar parte de un algoritmo. Además, se dice que una secuencia  de tareas está ‘bien definida’ si se sabe el orden exacto en que deben ejecutarse.  A lo largo de este libro, se considerará solo esta definición informal de algoritmo. En  matemáticas se usa una definición formal que está fuera del alcance de este texto.  EJECUTAR un algoritmo consiste en realizar las tareas o instrucciones que lo conforman, en el  orden especificado y utilizando los recursos disponibles. Hoy se cuenta con máquinas que  realizan esta labor, pero se requiere que los algoritmos que ejecutan se escriban en un lenguaje  especial. Usar esos lenguajes especiales para especificar algoritmos se llama programación de  computadores.  **2.3.1. *Características de un algoritmo***  Las características que debe poseer una secuencia de tareas para considerarse algoritmo son:  precisión, de finitud y finitud.  14  Precisión De finitud o Determinismo Finitud  Hay un orden preciso en el cual  deben ejecutarse las tareas que  conforman el algoritmo.  Todas las veces que se realicen  las tareas o pasos de un  algoritmo, con las mismas  condiciones iniciales, se deben  obtener resultados idénticos.  El algoritmo debe terminar en  algún momento y debe usar una  cantidad finita de recursos.  **2.3.2. *Estructura básica de un algoritmo***  En esencia un algoritmo está constituido por los siguientes tres elementos:  Datos Instrucciones Estructuras de control  Para almacenar información:  datos de entrada, de salida o  intermedios.  Las acciones o procesos que  el algoritmo realiza sobre los  datos.  Las que determinan el orden en que se  ejecutarán las instrucciones del  algoritmo.  En el capítulo siguiente se presentan de manera completa los conceptos de dato e instrucción, y  en el capítulo cuatro se describe el concepto de estructura de control.  **2.3.3. *Ejemplos de algoritmos***  Enseguida se dan varios ejemplos de algoritmos, algunos de los cuales no son susceptibles de  ejecutarse por medio de un computador; en cambio, son más bien recetas que se usan para  resolver problemas cotidianos.  ***PROBLEMA UNO:*** Un estudiante se encuentra en su casa (durmiendo) y debe ir a la universidad  (a tomar la clase de programación!), ¿qué debe hacer?  **ALGORITMO:**  **Inicio**  **PASO 1.** Dormir  **PASO 2. Hacer** 1 **hasta** que suene el despertador (o lo llame la mamá).  **PASO 3.** Mirar la hora.  **PASO 4.** ¿Hay tiempo suficiente?  **PASO 4.1. Si** hay, **entonces**  **PASO 4.1.1.** Bañarse.  **PASO 4.1.2.** Vestirse.  **PASO 4.1.3.** Desayunar.  **PASO 4.2. Sino,**  **PASO 4.2.1.** Vestirse.  **PASO 5.** Cepillarse los dientes.  **PASO 6.** Despedirse de la mamá y el papá.  **PASO 7.** ¿Hay tiempo suficiente?  **PASO 7.1. Si** hay, **entonces**  **PASO 7.1.1.** Caminar al paradero.  **PASO 7.2. Sino**, Correr al paradero.  **PASO 8. Hasta** que pase un bus para la universidad **hacer**:  **PASO 8.1.** Esperar el bus  **PASO 8.2.** Ver a las demás personas que esperan un bus.  **PASO 9.** Tomar el bus.  **PASO 10. Mientras** no llegue a la universidad **hacer**:  **PASO 10.1.** Seguir en el bus.  **PASO 10.2.** Pelear mentalmente con el conductor.  **PASO 11.** Timbrar.  **PASO 12.** Bajarse.  **PASO 13.** Entrar a la universidad.  **Fin**  15  ***PROBLEMA DOS:*** Sean *P=(a,b)* y *Q=(c,d)* los puntos extremos de un segmento de recta.  Encontrar un segmento de recta perpendicular al anterior, que pase por su punto medio.  **ALGORITMO:**  **Inicio**  **PASO 1.** Trazar un círculo con centro en el punto P que pase por el punto Q.  **PASO 2.** Trazar un círculo con centro en el punto Q que pase por el punto P.  **PASO 3.** Trazar un segmento de recta entre los puntos de intersección de las circunferencias  trazadas en los pasos 1 y 2.  **Fin.** El segmento de recta trazada es el buscado.  16  ***PROBLEMA TRES:*** Realizar la suma de dos números enteros positivos.  **ALGORITMO:**  **Inicio**  **PASO 1.** En una hoja de papel, escribir los números el primero arriba del segundo, de tal manera que las  unidades, decenas, centenas, etc., de los números queden alineadas. Trazar una línea debajo del segundo  número.  **PASO 2.** Empezar por la columna más a la derecha.  **PASO 3.** Sumar los dígitos de dicha columna.  **PASO 4.** Si la suma es mayor a 9 anotar el número que corresponde a las decenas encima de la siguiente  columna a la izquierda y anotar debajo de la línea las unidades de la suma. Si no es mayor anotar la suma debajo  de la línea.  **PASO 5.** Si hay más columnas a la izquierda, pasar a la siguiente columna a la izquierda y volver a 3.  **PASO 6.** El número debajo de la línea es la solución.  **Fin**  ***PROBLEMA CUATRO:*** Cambiar la rueda pinchada de un automóvil teniendo un gato mecánico  en buen estado, una rueda de reemplazo y una llave inglesa.  **ALGORITMO:**  **Inicio**  **PASO 1.** Aflojar los tornillos de la rueda pinchada con la llave inglesa.  **PASO 2.** Ubicar el gato mecánico en su sitio.  **PASO 3.** Levantar el gato hasta que la rueda pinchada pueda girar libremente.  **PASO 4.** Quitar los tornillos y la rueda pinchada.  **PASO 5.** Poner rueda de repuesto y los tornillos.  **PASO 6**. Bajar el gato hasta que se pueda liberar.  **PASO 7.** Sacar el gato de su sitio.  **PASO 8.** Apretar los tornillos con la llave inglesa.  **Fin**  ***PROBLEMA CINCO:*** Encontrar los números primos entre 1 y 50.  **ALGORITMO:**  **Inicio**  **PASO 1.** Escribir los números de 1 al 50  **PASO 2.** Tachar el número 1 ya que no es primo.  **PASO 3.** Para k entre 2 y el entero más cercano por debajo de la raíz cuadrada de 50, si  el número k no esta tachado, tachar los múltiplos del número k, sin tachar el número k.  **PASO 4.** Los números que no se tacharon son los números primos entre 1 y 50.  **Fin**  17  **2.3.4. *Representación de Algoritmos***  Cuando se quiere que un computador ejecute un algoritmo es indispensable, por lo menos hasta  hoy, representar ese algoritmo mediante algún formalismo. Las técnicas utilizadas más  comúnmente para la representación de algoritmos son:  2.3.4.1. **A Diagramas de flujo**  Se basan en la utilización de diversos símbolos geométricos para representar operaciones  específicas. Se les llama diagramas de flujo porque los símbolos utilizados se conectan por  medio de flechas para indicar la secuencia que sigue la ejecución de las operaciones.  En la sección de Programación Estructurada se explica la forma de construir diagramas de flujo  utilizando esta técnica.  *2.3.4.2.* ***Pseudocódigo***  Es un lenguaje de especificación de algoritmos. Tiene asociado un léxico (conjunto de palabras),  una sintaxis (reglas gramaticales) y una semántica precisa (significado), de manera análoga a  un lenguaje natural como el castellano. El uso de tal lenguaje hace relativamente fácil el paso de  codificación final del algoritmo (esto es, la traducción a un lenguaje de programación).  La ventaja del pseudocódigo es que le permite al programador concentrarse en la lógica y en las  estructuras de control del algoritmo que quiere diseñar, sin preocuparse de las reglas de un  lenguaje específico de programación, que normalmente incluyen infinidad de detalles. Es  también fácil modificar el pseudocódigo si se descubren errores o anomalías en la lógica del  programa; además de lo anterior, es fácil su traducción a lenguajes de programación como  *Pascal*, *C, Java* o *Basic*.  El pseudocódigo utiliza palabras reservadas (similares a sus homónimos en los lenguajes de  programación) para representar las acciones y estructuras de control, tales como *inicio*, *fin*, *sientonces-*  *sino*, *mientras*, etc.  El formato general de un algoritmo expresado mediante pseudocódigo es el siguiente:  [<definición de registros o tipos de datos>]  [**constantes** <declaración constantes>]  [**variables** <declaración variables globales del programador>]  [<definición de funciones y procedimientos>]  **procedimiento principal()**  [**constantes** <declaración constantes>]  [**variables** <declaración variables>]  **inicio**  /\*bloque de instrucciones\*/  **fin\_procedimiento**  Los detalles completos de este formato no son importantes a esta altura del libro y cada uno de  ellos será cubierto adecuadamente en próximos capítulos. Por ahora basta con saber que en las  primeras secciones se definen las constantes y variables que se usan dentro del algoritmo;  posteriormente se especifican algunos procedimientos y funciones; y finalmente, se escriben las  instrucciones que conforman el algoritmo principal.  También conviene mencionar que la información que se manipula dentro de un algoritmo se  agrupa en conjuntos llamados TIPOS. Hay unos tipos predefinidos en el pseudocódigo: **entero**  (conjunto de valores enteros), **real** (conjunto de valores reales), **caracter** (conjunto de  18  símbolos representables en el computador) y **booleano** (los dos valores lógicos: falso y  verdadero).  **PROGRAMACIÓN ESTRUCTURADA**  La programación estructurada es un estilo de programación de los años sesentas en el cual, la  estructura de un algoritmo se hace tan clara como sea posible utilizando tres formas de  organizar sus instrucciones:  1. Secuencia  2. Selección  3. Iteración  Estos tres tipos de estructuras de control pueden combinarse para producir programas de  computador tan complejos y largos como se quiera.  La lógica y propósito de un algoritmo estructurado puede comprenderse al leerlo de arriba hacia  abajo. Esto facilita que el creador de un programa lo comparta con otros programadores, lo dé a  conocer, o delegue en otros la labor de modificarlo.  Un programa (o algoritmo) estructurado tiene segmentos o componentes claramente definidos.  Además, en cada componente está explícito un punto de inicio o entrada, y un punto de  finalización o salida. Tal segmento de denomina un ***programa propio***.  **2.5.1. *Teoría de la Programación Estructurada***  ***Teorema de la estructura.***  Cualquier programa propio se puede escribir usando solamente las tres estructuras de control:  secuencia, selección e iteración.  Un programa propio contempla dos propiedades básicas:  1. Tiene exactamente un punto de entrada y uno de salida  2. Entre el punto de entrada y el de salida hay trayectorias que conducen a cada parte del  programa; esto significa que no existen grupos de instrucciones que se ejecuten  indefinidamente, ni instrucciones que jamás se vayan a ejecutar.  Las tres estructuras de control se ilustran a continuación.  *Secuencia*: Las instrucciones del programa se ejecutan en el orden en el cual ellas aparecen en  el texto del programa como se indica en el siguiente diagrama:  A B A B  A y B pueden ser instrucciones básicas o compuestas. A y B deben ser ambos programas propios  en el sentido ya definido de entrada y salida. La combinación de A y B es también un programa  propio y que tiene también una entrada y una salida.  22  *Selección*: permite escoger entre dos grupos de instrucciones, de acuerdo a la evaluación de una  condición o predicado lógico. Se conoce como estructura SI–ENTONCES–SINO. En el siguiente  diagrama, P representa un predicado y, A y B representan grupos de instrucciones. Debe ser  claro que si P es falso, se realiza el conjunto de instrucciones B; de lo contrario se realiza A. Las  flechas horizontales de más a la izquierda y de más a la derecha representan los puntos de  entrada y salida, respectivamente.  A  V  P  F  B  *Iteración*: Esta forma de control permite repetir varias veces una instrucción o conjunto de  instrucciones hasta cuando deje de cumplirse una condición (predicado lógico).  Se conoce como la estructura HACER – MIENTRAS. En el siguiente diagrama, P representa un  predicado y A representa un grupo de instrucciones. Debe ser claro que siempre que P sea  verdadero, se realiza el conjunto de instrucciones A; de lo contrario se termina la iteración. Las  flechas horizontales de más a la izquierda y de más a la derecha representan los puntos de  entrada y salida, respectivamente.  A  V  F  P  Algunos de los símbolos utilizados en las estructuras para conformar los diagramas de flujo son:  Representa el inicio y el fin de un programa  Entrada de datos(lectura)  Salida de datos (impresión)  Proceso (cualquier tipo de operación)  23  Indicador de dirección o flujo  **2.6. EJERCICIOS**  **2.6.1. *Ejercicios de Problemas***  Para los siguientes problemas, determine las variables conocidas, desconocidas, las condiciones  y el tipo de problema. Para aquellos problemas algorítmicos desarrollar adicionalmente un  algoritmo que permita encontrar una solución.  1. Se tienen dos jarras (*A* y *B*) de capacidades 3 y 7 litros respectivamente, sobre las cuales se  pueden efectuar las siguientes acciones: Llenar totalmente cualquiera de las dos jarras, vaciar  una de las dos jarras en la otra hasta que la jarra origen este vacía o hasta que la jarra destino  este llena y vaciar el contenido de una jarra (este llena o no) en un sifón. ¿Cómo se puede dejar  en la jarra A un solo litro utilizando solamente las anteriores acciones?  2. Es cierta o no es cierta la siguiente frase: “Esta frase no es cierta”.  3. Si Juan tiene el doble de la edad de Pedro y la suma de las edades de los dos es 33 años,  ¿Cuántos años tiene Juan y cuántos tiene Pedro?  4. ¿Qué figura se forma al unir los puntos marcados con números consecutivos con una línea?  5. Calcular de manera exacta el número de átomos del universo.  6. Calcular el costo de una serie de productos comprados en el supermercado.  7. Determinar quien es el mejor jugador de fútbol de toda la historia.  8. Construir un barco de papel.  **2.6.2. *Ejercicios de algoritmos***  Para los siguientes problemas construir un algoritmo que los solucione.  ***EJERCICIO UNO:*** Buscar en el directorio telefónico, el número de:  a. José González Pérez  b. Pedro Gómez Bernal.  c. Escribir un algoritmo que sirva para buscar a cualquier persona.  ***EJERCICIO DOS:*** Calcular el número de días entre las fechas:  a. Enero 17 de 1972 y Julio 20 de 1973  b. Febrero 2 de 1948 y Agosto 11 de 1966  24  c. Escribir un algoritmo que sirva para calcular la cantidad de días entre dos fechas  cualesquiera.  ***EJERCICIO TRES:*** Solicitar en préstamo algún libro de una biblioteca.  ***EJERCICIO CUATRO:*** Hacer una caja de cartón con tapa de:  a. 20 cm de largo, por 10 cm de ancho y 5 cm de alto.  b. 10 cm de largo, por 30 cm de ancho y 15 cm de alto.  c. Escribir un algoritmo que sirva para construir una caja de cartón con tapa de  cualquier tamaño.  ***EJERCICIO CINCO:*** Construir un avión de papel.  ***EJERCICIO SEIS:*** Calcular manualmente la división de cualquier par de números naturales. El  resultado también debe ser un número natural. Escribir un algoritmo para calcular el residuo de  la división.  ***EJERCICIO SIETE:*** Un juego muy famoso entre dos niños es el de *adivina mi número*, el cual  consiste en que cada niño trata de adivinar el número pensado por el otro niño. Dicho número  generalmente está entre 1 y 100. Las reglas del juego son las siguientes:  a. Cada niño posee un turno en el que trata de averiguar el número del otro.  b. En su turno el primer niño pregunta si un número que dice es el pensado por el  segundo.  c. Si el número que ha dicho el primer niño es el que pensó el segundo, este último  debe informarle al primero que ganó.  d. Si el número no es el segundo niño debe decir si su número pensado es menor o  mayor al que el primer niño dijo.  e. Luego el segundo niño tiene su turno y de esta manera se van intercalando hasta  que alguno de los dos gane. Desarrollar un algoritmo para jugar adivina mi número.  ***EJERCICIO OCHO:*** Una balanza se encuentra en equilibrio cuando el producto de la carga  aplicada sobre el brazo derecho por la longitud de este brazo, es igual al producto de la carga  aplicada sobre el brazo izquierdo por la longitud de este otro brazo. Determinar si la balanza se  encuentra en equilibrio si:  a. La longitud del brazo izquierdo es 3 m, la del derecho es 2 m, la carga aplicada al  brazo izquierdo es 5 Kg y la carga aplicada al derecho es 7 Kg.  b. La longitud del brazo izquierdo es 4 m, la del derecho es 2 m, la carga aplicada al  brazo izquierdo es 4 Kg y la carga aplicada al derecho es 4 Kg.  c. Desarrollar un algoritmo que sirva para cualquier conjunto de valores para las  longitudes de los brazos y las cargas aplicadas.  ***EJERCICIO NUEVE*:** Pasar un número entero positivo de base a diez a binario. Hacer también  un algoritmo que haga la transformación contraria.  25  **3. METODOLOGÍA DE SOLUCIÓN DE PROBLEMAS CON EL COMPUTADOR**  **3.1. OBJETIVO**  El objetivo de este capítulo es : lograr que el lector entienda cuál es la metodología que se usará  en la solución de los problemas. Para esto, se explica primero cuáles son los pasos de la  metodología, y luego se describen algunos ejemplos de su aplicación a pequeños problemas de  programación.  **3.2. INTRODUCCIÓN**  A través del tiempo, y mediante la práctica constante, se ha desarrollado en ingeniería un  conjunto muy amplio de métodos o formas estructuradas de proceder ante ciertas clases de  problemas que se presentan con frecuencia. Este conjunto de métodos disponibles para afrontar  un problema se llama metodología.  En ingeniería de sistemas se han venido depurando varias metodologías para programar  computadores de manera adecuada y efectiva. Estas metodologías no se deben suponer como  fórmulas mágicas, sino más bien como formas de proceder que, seguidas con disciplina,  conducen a buenos resultados, en la mayoría de los casos. Programar sin metodología se  traduce en invertir mucho más tiempo del necesario, en sufrir muchos dolores de cabeza y muy  seguramente, en obtener resultados de baja calidad.  El desarrollo de un programa que resuelva un problema es generalmente un reto intelectual  importante. Desde hace más o menos 50 años se están haciendo programas para resolver  problemas de diversa índole. Este ejercicio continuado y prolífico ha dado lugar a la formulación  de *metodologías de programación*. En este libro se sigue una metodología sencilla que se espera  que el estudiante adopte como forma de proceder para afrontar la mayoría de los problemas de  programación a que se enfrente.  Esta metodología es un conjunto o sistema de métodos, principios y reglas que permiten  enfrentar de manera sistemática el desarrollo de un programa que resuelva un problema  algorítmico. Esta metodología se estructura como una secuencia de pasos que parten de la  definición del problema y culminan con un programa que lo resuelve.  A continuación se presentan de manera general los pasos de una metodología:  **Análisis del problema** Con la cual se busca comprender totalmente el problema a resolver.  **Especificación del problema**  Con la cual se establece de manera precisa cuáles son los datos de  entrada, los de salida y qué condiciones debe cumplir cada uno de ellos.  **Diseño del algoritmo**  En esta etapa se construye un algoritmo que cumpla con la especificación  formulada en el paso anterior  Problema  Metodología  Programa  26  **Prueba del algoritmo y**  **refinamiento**  Consiste en comprobar experimentalmente que el algoritmo funciona bien y  corregir los posibles errores que se detecten.  **Codificación** Se traduce el algoritmo a un lenguaje de programación.  **Prueba y Verificación**  Se realizan pruebas del programa implementado para determinar su  efectividad en la resolución del problema.  NOTA: LOS EJEMPLOS INCLUIDOS EN ESTA SECCIÓN EN PSEUDICÓDIGO SON SOLO PARA ILUSTRAR LA  METODOLOGÍA, NO SE ESPERA QUE EL ESTUDIANTE LOS COMPRENDA  **3.3. ANÁLISIS DEL PROBLEMA**  Es una perogrullada, pero hay que decirla: **antes de poder dar solución adecuada a un**  **problema hay que entenderlo completamente**. En el primer paso de la metodología el  programador debe alcanzar claridad acerca de lo que le están pidiendo resolver. Concretamente,  debe determinar de manera clara y concisa lo siguiente:  1. *Los objetos, datos conocidos o datos de entrada*. Aquellos elementos de información  total o parcial que se encuentran en el enunciado del problema, y que son útiles en la  búsqueda de los objetos desconocidos.  2. *Los objetos, datos desconocidos o datos de salida*. Aquellos datos que el algoritmo debe  entregar como solución, al final de su ejecución.  3. *Las condiciones*. De una parte están las condiciones que deben cumplir los datos de  entrada. De otra parte están las propiedades que deberán cumplir los datos de salida,  que son normalmente relaciones con los datos de entrada. Estas relaciones establecen  una dependencia de los datos de entrada.  **Ejemplo.** Sean los puntos *P=(a,b)* y *Q=(c,d)* que definen una recta, encontrar un segmento de  recta perpendicular a la anterior que pase por el punto medio de los puntos dados.  **OBJETOS CONOCIDOS** Los puntos *P* y *Q*.  **OBJETOS DESCONOCIDOS** Un segmento de recta  **CONDICIONES**  Los puntos P y Q son diferentes  El segmento de recta pedido debe pasar por el punto medio entre *P* y *Q*, y  debe ser perpendicular a la recta trazada entre *P* y *Q*.  **3.4. ESPECIFICACION DEL PROBLEMA**  Después de entender totalmente el problema a resolver (lo cual se consigue con la etapa de  análisis), se debe realizar su especificación que corresponde a una formalización del análisis. La  especificación de un problema se hace mediante una descripción clara y precisa de:  1. Las entradas que recibirá el algoritmo pedido. Las entradas corresponden a los objetos  conocidos. Se debe indicar la descripción, cantidad y tipo de las mismas.  2. Las salidas que el algoritmo pedido proporcionará. Las salidas corresponden a los objetos  desconocidos del problema. Se debe indicar la cantidad, descripción y tipo de las  mismas.  3. La dependencia que mantendrán las salidas obtenidas con las entradas recibidas. Por  una parte, se debe hacer claridad sobre las condiciones o propiedades que deben cumplir  los datos de entrada. De otro lado, se describe claramente como dependen las salidas de  las entradas.  Esta descripción puede estar acompañada de un diagrama de caja negra como el de la siguiente  figura. En ella, a cada entrada y salida se le pone un nombre adecuado y el algoritmo pedido se  27  representa con una caja, para indicar que no se conoce, por ahora, nada acerca de su estructura  interna.  **Ejemplo 1.** Construir un algoritmo que calcule el promedio de 4 notas.  **ESPECIFICACION:**  **Entradas** N1,N2,N3,N4 (notas parciales) de tipo Real.  **Salidas** Final (nota final) de tipo Real.  **Condiciones**  N1,N2,N3,N4 deben ser números que corresponden a notas válidas  **DIAGRAMA DE CAJA NEGRA:**  **Ejemplo 2.** Construir un algoritmo que determine el mayor de tres números enteros dados.  **ESPECIFICACIÓN:**  **Entradas** A,B y C. Tres números de tipo Real.  **Salidas** Mayor (valor mayor) de tipo Real.  **Condiciones**  Mayor debe ser el valor máximo de A, B y C.  Formalmente:  28  **DIAGRAMA DE CAJA NEGRA:**  **Ejemplo 3.** Determinar si un punto está dentro de un círculo.  **ESPECIFICACIÓN:**  **Entradas**  Cx (abscisa del centro del círculo) de tipo Real.  Cy (ordenada del centro del círculo) de tipo Real.  r (radio del círculo) de tipo Real.  Px (coordenada x del punto) de tipo Real.  Py (coordenada y del punto) de tipo Real.  **Salidas**  ***Pertenece*** de tipo Booleano, (indica si el punto está dentro o fuera del  círculo).  **Condiciones**  r>0 (**ojo!**: condición sobre un dato de entrada)  Pertenece = Verdadero, si el punto está dentro del círculo.  Pertenece = Falso, si el punto está fuera del círculo.  Formalmente:  **DIAGRAMA DE CAJA NEGRA:**  **3.5. DISEÑO ESTRUCTURADO DE ALGORITMOS**  En la fase de diseño del algoritmo se especifica un conjunto de instrucciones capaz de obtener  datos de salida correctos, a partir de datos válidos de entrada. Normalmente, un algoritmo tiene  una estructura no trivial, es decir, pueden identificarse en él, partes o componentes muy  claramente delimitados que interactúan durante la ejecución. Una primera estrategia para  diseñar el algoritmo que solucione el problema consiste en identificar los componentes de esa  29  estructura, para luego diseñarlos individualmente, y por último ensamblarlos en el programa  completo. Esta forma de proceder se denomina *diseño top-down* porque consiste en solucionar  un problema dividiéndolo en otros más pequeños, solucionando esos problemas resultantes y  finalmente, ensamblando las soluciones de esos subproblemas para obtener la solución completa  del problema original.  **3.5.1. *División***  Consiste en identificar subprocesos dentro del proceso completo que debe llevar a cabo el  algoritmo buscado. Está subdivisión se realiza de manera repetida hasta llegar al nivel de  instrucción básica. La siguiente figura ilustra el modo de proceder.  En la etapa final de subdivisión, los problemas a resolver son tan sencillos que ya no se pueden  dividir más, sino que simplemente se pueden resolver usando las instrucciones básicas  disponibles en el pseudocódigo: asignación, lectura o escritura, secuencia, selección, repetición.  **Problema**: Realizar un programa que lea una serie de *n* números enteros y determine si la  suma de los mismos es un cuadrado perfecto.  **ESPECIFICACIÓN DEL PROBLEMA**  **Entradas**  n (número de datos a considerar)  dato1, dato2, ..., daton (los números enteros)  **Salidas**  ***Es\_cuadrado*** de tipo Booleano, (indica si la suma de los datos de entrada  es un cuadrado perfecto).  **Condiciones**  n>0  **Es\_cuadrado** tiene valor verdadero si la suma de los n números es un  cuadrado perfecto. De lo contrario, tendrá valor falso.  Diagrama de caja negra:  30  **DISEÑO DEL ALGORITMO:**  El algoritmo que soluciona este problema es muy sencillo. Sin embargo, para el propósito de  ilustrar la técnica de subdivisiones sucesivas (diseño top-down o refinamiento a pasos) se  realizan tres etapas de división, cada una de las cuales tiene un diagrama de flujo asociado.  En la primera fase de división se considera que el algoritmo consiste, grosso modo, en leer los  datos de entrada, sumar los n números, y determinar si el resultado de la suma es un cuadrado  perfecto.  Primera Iteración  En la segunda fase de división se hace explícito que el proceso de lectura de los datos consiste  en **(i)** obtener el número de enteros que se van a procesar y en **(ii)** leer cada uno estos n  enteros. Adicionalmente, se aclara de qué manera se verificará que la suma es un cuadrado  perfecto (se comprueba si la raíz cuadrado es un entero). Por último se deja especificado lo que  el algoritmo escribirá como resultado final.  31  Segunda Iteración  En la tercera fase de división se logra describir todo las acciones del algoritmo con instrucciones  básicas. En este punto ya no es necesario ni posible hacer más subdivisiones. El algoritmo está  diseñado completamente.  Aunque el lector no entienda todos los detalles del lenguaje con el cual se describe el algoritmo,  se le invita a entender la lógica del mismo.  32  Tercera Iteración (final en este ejemplo)  33  En seguida se muestra la traducción del diagrama de flujo anterior a pseudocódigo. Nuevamente  se aclara que los detalles de esta forma de describir algoritmos se estudian en los próximos  capítulos.  **Pseudocódigo**  **procedimiento principal()**  **variables**  **n : entero**  **suma : entero**  **dato : entero**  **i : entero**  **inicio**  **escribir ( “Número de enteros a considerar:”)**  **leer( n)**  **suma := 0**  **Para( i:=1 hasta n) hacer**  **escribir(“ingrese un número entero:” )**  **leer( dato)**  **suma := suma + dato**  **fin\_para**  **si(piso( raiz2( suma ) ) = raiz2(suma ))**  **Entonces**  **escribir (“La suma de los números es un cuadrado perfecto”)**  **sino**  **escribir (“La suma de los números no es un cuadrado perfecto”)**  **fin\_si**  **fin\_procedimiento**  **3.5.2. *Definición de abstracciones***  Durante el proceso de división es posible identificar qué secuencias de pasos se utilizan más de  una vez en diferentes partes del proceso completo. Los lenguajes de programación brindan la  posibilidad de escribir solamente una vez estas secuencias de pasos, mediante el uso de unas  **abstracciones o subprogramas** llamados procedimientos y funciones. Lo que debe hacer el  programador es:  • Recolectar estas secuencias de pasos en funciones y procedimientos, según sea el caso.  • Documentar cada función y procedimiento especificando claramente:  o El propósito de la función (o procedimiento).  o El nombre, tipo y propósito de cada argumento.  o El resultado que produce ese subprograma (o efectos laterales).  El uso de funciones y procedimientos evita tener que escribir más de una vez las secuencias de  pasos que se repiten. Pero lo más importante es que permiten describir más claramente la lógica  del programa.  **Problema:** Desarrollar un programa que determine si un número dado se encuentra en alguno  de dos conjuntos finitos de enteros.  Recuerde que lo primero que recomienda la metodología propuesta es entender completamente  el problema y hacer su especificación.  En este ejercicio se usan arreglos para guardar los dos conjuntos de números. El uso de arreglos  para representar conjuntos, requiere que se realicen ciertas validaciones que garanticen la  validez como conjunto, por ejemplo, que en el arreglo no esté un mismo elemento dos veces (en  un conjunto un elemento está solo una vez). Los arreglos son un tipo de estructura de datos que  34  se cubre en un capítulo posterior. Teniendo en cuenta estos razonamientos, se puede especificar  el problema:  **ESPECIFICACIÓN DEL PROBLEMA**  **Entradas** dos conjuntos, A y B; y el número entero.  **Salidas**  ***bandera*** de tipo Booleano, (indica si el número está o no en  alguno de los dos conjuntos).  **Condiciones**  si el entero dado está en alguno de los conjuntos dados el valor de  verdad de ***bandera*** será verdadero. Será falso en caso contrario.  Diagrama de caja negra:  donde,  A: es un conjunto de enteros.  B: es un conjunto de enteros.  elemento: es el entero a comprobar si está en alguno de los conjuntos.  bandera : es un booleano que indica si el elemento está o no está.  **Diseño del algoritmo**  Primera iteración. De manera general, el algoritmo debería tener los siguientes pasos  **Inicio**  **1.** Leer los dos conjuntos.  **2.** Leer entero.  **3.** Determinar si el entero está en alguno de los dos conjuntos  **4.** Imprimir el resultado.  **Fin**  Segunda iteración. En esta segunda fase se aclara en qué consiste leer los conjuntos.  **1.** Leer primer conjunto consiste en:  **1.1.** Leer un dato.  **1.2.** Determinar si el elemento no está en el primer conjunto.  **1.3.** Si no está, agregar el dato al primer conjunto. Si ya está, mostrar un mensaje de error.  **1.4.** Preguntar si el usuario desea ingresar un nuevo elemento al primer conjunto.  **1.5.** Si el usuario desea ingresar un nuevo elemento volver a 1.1. Si no, continuar.  **2.** Leer segundo conjunto se divide en:  35  **2.1.** Leer un dato.  **2.2.** Determinar si el elemento no está en el segundo conjunto.  **2.3.** Si no está, agregar el dato al segundo conjunto. Si ya está, mostrar un mensaje de error.  **2.4.** Preguntar si el usuario desea ingresar un nuevo elemento al segundo conjunto.  **2.5.** Si el usuario desea ingresar un nuevo elemento volver a 2.1. Sino, continuar.  **3.** Leer entero.  **4.** Determinar si el entero está en el primer conjunto.  **5.** Determinar si el entero está al segundo conjunto.  **6.** Imprimir el resultado.  Tercera Iteración. Se utiliza pseudocódigo para especificar todas las operaciones necesarias.  **procedimiento principal()**  **variables**  **i, j, n, m, elemento : entero**  **continuar : carácter**  **bandera : booleano**  **A : arreglo [100] de entero**  **B : arreglo [100] de entero**  **/\* el codigo siguiente lee el conjunto A\*/**  **Inicio**  **n := 0**  **escribir(“Desea ingresar elementos al conjunto A (S/N):”)**  **leer( continuar)**  **mientras( continuar = ‘S’ | continuar = ‘s’) hacer**  **escribir( “Ingrese el elemento al conjunto A:”)**  **leer( elemento)**  **/\* el codigo siguiente prueba si el elemento esta en el conjunto A \*/**  **i := 0**  **mientras( i<n & A[i]<> elemento) hacer**  **i := i+1**  **fin\_mientras**  **si( i = n) entonces**  **A[n] := elemento**  **n := n+1**  **Sino**  **escribir( “Error: el elemento ya esta en el conjunto A” )**  **fin\_si**  **escribir( “Desea ingresar mas elementos al conjunto A (S/N)” )**  **leer( continuar)**  **fin\_mientras**  **/\* el codigo siguiente lee el conjunto B \*/**  **m := 0**  **escribir(“Desea ingresar mas elementos al conjunto B (S/N)” )**  **leer( continuar)**  **mientras( continuar = ‘S’ | continuar = ‘s’) hacer**  **escribir( “Ingrese el elemento al conjunto B:”)**  **leer( elemento)**  **/\* el codigo siguiente prueba si el elemento esta en el conjunto B \*/**  **i := 0**  **mientras i<m & B[i] <> elemento hacer**  **i := i+1**  **fin\_mientras**  **si (i = m) entonces**  **B[m] := elemento**  **m := m+1**  **sino**  **escribir( “Error: el elemento ya esta en el conjunto B” )**  **fin\_si**  **escribir( “Desea ingresar mas elementos al conjunto B (S/N)” )**  36  **leer( continuar)**  **fin\_mientras**  **/\* el codigo siguiente lee un elemento a probar \*/**  **escribir( “Ingrese el dato que desea buscar en los conjuntos” )**  **leer( elemento)**  **/\* el codigo siguiente prueba si el elemento esta en el conjunto A \*/**  **bandera := falso**  **i := 0**  **mientras( i<n & A[i] <> elemento) hacer**  **i := i+1**  **fin\_mientras**  **si( i < n) entonces**  **bandera := verdadero**  **fin\_si**  **/\* el codigo siguiente prueba si el elemento esta en el conjunto B \*/**  **i := 0**  **mientras( i<m & B[i] <> elemento) hacer**  **i := i+1**  **fin\_mientras**  **si(i < m) entonces**  **bandera := verdadero**  **fin\_si**  **/\* el codigo siguiente determina si el elemento esta en alguno de los dos conjuntos \*/**  **si(bandera = verdadero) entonces**  **escribir( “El dato dado esta en alguno de los dos conjuntos” )**  **sino**  **escribir( “El dato dado NO esta en ninguno de los conjuntos” )**  **fin\_si**  **fin\_procedimiento**  **Abstracción:** En el código obtenido mediante la fase de división se puede apreciar la existencia  de porciones de código que no son idénticas, pero son muy parecidas y funcionalmente  equivalentes. Este es el caso de las porciones de código que permiten leer los conjuntos A y B, y  las porciones de código que permiten determinar si un elemento está en el conjunto A y en el  conjunto B. Identificadas estas redundancias, se puede crear un procedimiento que permita leer  un conjunto cualquiera, y una función que verifique si un elemento está en un conjunto dado.  Primero, la función **pertenece** es un subprograma que se define así:  pertenece: **Arreglo**[100] de **Entero** x **Entero** x **Entero** -> **Booleano**  *( A , , n , e ) ÞV si e = A[i] para algún i*  *Þ F en otro caso*  Se puede observar que esta función además de recibir el arreglo de datos y el elemento, recibe  un entero adicional *n*. Este entero se utiliza para indicar el tamaño del conjunto dado. Esta  función se codifica como sigue:  **funcion** pertenece( A :**arreglo**[100] de **entero**, n :**entero**, e :**entero** ):**booleano**  variables  bandera :**booleano**  i : **entero**  inicio  /\* el codigo siguiente prueba si el elemento está en el conjunto \*/  i := 0  **mientras(** i<n & A[i] <> e**) hacer**  i := i+1  **fin\_mientras**  **si(** i = n**) entonces**  bandera := falso  37  **si\_no**  bandera := verdadero  **fin\_si**  **retornar** bandera  **fin\_funcion**  Segundo, el procedimiento **leer\_conjunto** es un subprograma que se define así:  **Procedimiento** leer\_conjunto( **var** A :**arreglo** [100] de **entero**, **var** n :**entero**)  **variables**  i :**entero**  elemento :**entero**  continuar :**caracter**  **inicio**  /\* el codigo siguiente lee el conjunto \*/  n := 0  **escribir(** “Desea ingresar mas elementos al conjunto (S/N)?”)  **leer(** continuar)  **mientras ( n<100 &** (continuar = ‘S’ | continuar = ‘s’)) **hacer**  **escribir(** “Ingrese el elemento al conjunto ” )  **leer(** elemento)  /\* el código siguiente prueba si el elemento esta en el conjunto y de no ser así lo adiciona \*/  **si** no (pertenece( A, n, elemento) ) **entonces**  A[n] := elemento  n := n+1  **sino**  **escribir(** “Error: el elemento ya esta en el conjunto ” )  **fin\_si**  **escribir(** “Desea ingresar mas elementos al conjunto (S/N)?”)  **leer(** continuar)  **fin\_mientras**  **fin\_procedimiento**  Este procedimiento recibe un arreglo de enteros y una variable entera llamada **n**. Lo que hace es  leer unos números (máximo 100) y guardarlos en el arreglo. También lleva la cuenta de cuántos  datos ha leído, y almacena esta cifra en la variable **n**.  Otro aspecto importante que se puede destacar en este procedimiento es que usa la función  ***pertenece*** para determinar si se debe o no adicionar el elemento leído al conjunto.  De esta manera el algoritmo principal se puede presentar como sigue:  **procedimiento principal()**  **variables**  n :**entero**  m :**entero**  elemento :**entero**  A :**arreglo** [100] de **entero**  B :**arreglo** [100] de **entero**  **inicio**  /\* el codigo siguiente lee el conjunto A \*/  leer\_conjunto( A, n )  /\* el codigo siguiente lee el conjunto B \*/  leer\_conjunto ( B, m)  /\* el codigo siguiente determina si el elemento esta en alguno de los dos conjuntos \*/  **si** pertenece( A, n, elemento ) | pertenece( B, m, elemento ) **entonces**  **escribir(** “El dato dado esta en alguno de los dos conjuntos”)  **sino**  **escribir(** “El dato dado NO esta en los conjuntos dados” )  **fin\_si**  **fin\_procedimiento**  38  Se puede apreciar la reducción de líneas de código del programa y la facilidad de lectura de cada  uno de estos algoritmos (función, procedimiento y algoritmo principal), con respecto al algoritmo  inicial realizado sin abstracción. Se deja al lector la escritura del programa completo, utilizando  las reglas descritas en esta sección.  **3.6. CODIFICACION**  Cuando ya se ha diseñado completamente el algoritmo y se tiene escrito en algún esquema de  representación (pseudo-código o diagrama de flujo), el siguiente paso es codificarlo en el  lenguaje de programación definido para tal fin.  En este momento es cuando el programador interactúa con el computador mediante la  herramienta de software de que disponga para codificar en el lenguaje seleccionado.  **EJEMPLO.** Tómese como base el pseudocódigo desarrollado en la sección anterior. El programa  en C++ para este pseudocódigo sería:  **#include <iostream.h>**  **bool** pertenece( **int** A[100, **int** n, **int** e)  **{**  **bool** bandera;  **int** i;  /\* el codigo siguiente prueba si el elemento esta en el conjunto \*/  i = 0;  **while**(> i<n && A[i] != e ) **{**  i = i+1;  **};**  **if**( i == n ) **{**  bandera = false;  **}**  **else {**  bandera = true;  **};**  **return** bandera;  **}**  **void** leer\_conjunto( **int**A[100], **int** & n,)  **{**  **int** i;  **int** elemento;  **char** continuar;  /\* el codigo siguiente lee el conjunto \*/  n = 0;  **cout** << “Desea ingresar mas elementos al conjunto (S/N)”;  **cin** >> continuar;  **while**( continuar == ‘S’ || continuar == ‘s’ ) **{**  **cout** << “Ingrese el elemento que quiere agregar :”;  **cin** >> elemento;  /\* el codigo siguiente prueba si el elemento esta en el conjunto y de no ser así lo adiciona \*/  **if**( !pertenece( A, n, elemento ) ) **{**  A[n] = elemento;  n = n+1;  **}**  **else {**  **cout** << “Error: el elemento ya esta en el conjunto”;  39  **};**  **cout** << “Desea ingresar mas elementos al conjunto (S/N)” ;  **cin**>> continuar;  **};**  **}**  **void main()**  **{**  **int** n, m, elemento;  **int** A[100];  **int** B[100];  /\* el codigo siguiente lee el conjunto A \*/  leer\_conjunto( A, n);  /\* el codigo siguiente lee el conjunto B \*/  leer\_conjunto( B, m)  /\* el codigo siguiente determina si el elemento esta en alguno de los dos conjuntos \*/  **if**( pertenece(A, n, elemento) || pertenece(B, m, elemento) ) **{**  **cout** << “El dato dado esta en alguno de los dos conjuntos”;  **}**  **else {**  **cout**<< “El dato dado NO esta en los conjuntos dados”;  **};**  **}**  **3.6.1. *Prueba de escritorio***  La prueba de escritorio es una herramienta útil para entender qué hace un determinado  algoritmo, o para verificar que un algoritmo cumple con la especificación sin necesidad de  ejecutarlo.  Básicamente, una prueba de escritorio es una ejecución ‘a mano’ del algoritmo, por lo tanto se  debe llevar registro de los valores que va tomando cada una de las variables involucradas en el  mismo.  A continuación se muestra un ejemplo de prueba de escritorio del siguiente algoritmo:  **procedimiento principal()**  **variables**  suma, entrada, menor :entero  **inicio**  **leer(** entrada)  menor := entrada  suma:= 0  **mientras** (entrada <> 0) **hacer**  **si** (entrada < menor) **entonces**  menor =entrada  **fin\_si**  **suma:**= suma + entrada  **leer(** entrada**)**  **fin\_mientras**  **escribir( “**valor menor:**” )**  **escribir(** menor**)**  **escribir( “**Suma total:**”)**  **escribir(** suma**)**  **fin\_procedimiento**  40  INSTRUCCIÓN entrada menor suma Pantalla  **leer (**entrada) 10  menor := entrada 10  suma :=0 0  suma :=suma + entrada 10  **leer** (entrada) 7  menor := entrada 7  suma:=suma + entrada 17  **leer(** entrada) 9  suma :=suma + entrada 26  **leer** (entrada) 0  **escribir** (**“**valor menor:**”)** Valor Menor  **escribir** (menor) 7  **escribir** (**“**Suma:**”)** Suma:  **escribir** (suma)  Trabajo en equipo  Múltiples roles...   Gestores   Analistas   Diseñadores   Programadores   Probadores   Administradores de  sistemas  ...  Fundamentos de la programación: Computadoras y programación Página 8  Parque Jurásico  Luis Hernández Yáñez  *Esquema general*  Fundamentos de la programación: Computadoras y programación Página 9  Memoria  temporal  Almacenamiento  permanente  Dispositivos  de entrada  Teclado  Ratón  Escáner  Táctil  …  Dispositivos  de salida  Monitor  Impresora  Altavoz  …  Unidad Central de Proceso  *Central Processor Unit*  C.P.U.  Luis Hernández Yáñez  *La arquitectura de Von Neumann*  Fundamentos de la programación: Computadoras y programación Página 10  C.P.U. (Procesador)  Dispositivos de E/S  Memoria  A.L.U.  Unidad Aritmético‐Lógica  Unidad de Control  Una ALU de 2 bits (Wikipedia)  Luis Hernández Yáñez  *La memoria*  Fundamentos de la programación: Computadoras y programación Página 11  Memoria  Bus  de  datos  01  02  03  04  05  06  07  08  **. . .**  Dirección  Cada celda en una dirección  Celdas de 8 / 16 / 32 / 64 bits  Información volátil  1 Bit = 0 / 1  1 Byte = 8 bits = 1 carácter  1 Kilobyte (KB) = 1024 Bytes  1 Megabyte (MB) = 1024 KB  1 Gigabyte (GB) = 1024 MB  1 Terabyte (TB) = 1024 GB  1 Petabyte (PB) = 1024 TB  210 = 1024  1000  Luis Hernández Yáñez  Fundamentos de la programación: Computadoras y programación Página 12  Luis Hernández Yáñez  *Los procesadores trabajan con ceros y unos (bits)*  Unidad de memoria básica: *Byte* (8 bits)  (2 dígitos hexadecimales: 01011011  0101 1011  5B)  *Lenguaje máquina*  Códigos hexadecimales que representan instrucciones,  registros de la CPU, direcciones de memoria o datos  Instrucción *Significado*  A0 2F *Acceder a la celda de memoria 2F*  3E 01 *Copiarlo el registro 1 de la ALU*  A0 30 *Acceder a la celda de memoria 30*  3E 02 *Copiarlo en el registro 2 de la ALU*  1D *Sumar*  B3 31 *Guardar el resultado en la celda de memoria 31*  Fundamentos de la programación: Computadoras y programación Página 13  Lenguaje de bajo nivel  Dependiente de la máquina  Programación difícil  Luis Hernández Yáñez  Nemotécnicos para los códigos hexadecimales:  A0  READ 3E  REG 1D  ADD …  Mayor legibilidad:  READ 2F  REG 01  READ 30  REG 02  ADD  WRITE 31  Lenguaje de nivel medio  Fundamentos de la programación: Computadoras y programación Página 14  Código objeto  (lenguaje máquina)  Programa  ensamblador  Código fuente  (lenguaje ensamblador)  Luis Hernández Yáñez  Fundamentos de la programación: Computadoras y programación Página 15  Luis Hernández Yáñez   Más cercanos a los lenguajes natural y matemático  resultado = dato1 + dato2;   Mayor legibilidad, mayor facilidad de codificación   Estructuración de datos / abstracción procedimental  Fundamentos de la programación: Computadoras y programación Página 16  Luis Hernández Yáñez  *Traducción*  Compiladores:  Compilan y enlazan  programas completos  Intérpretes:  Compilan, enlazan  y ejecutan instrucción  a instrucción  Fundamentos de la programación: Computadoras y programación Página 17  Compilador  Enlazador  Código  objeto de  biblioteca  Código fuente  #include <iostream>  using namespace std;  int main()  {  cout << "Hola Mundo!" << endl;  return 0;  }  Código objeto 0100010100111010011100…  Programa  ejecutable  Para una arquitectura concreta  y un sistema operativo  Luis Hernández Yáñez  *Genealogía de lenguajes*  Fundamentos de la programación: Computadoras y programación Página 18  BASIC  1964  Logo  1968  Ruby  1993  Python  1991  Eiffel  1986  Modula  1975  Fuente:  http://www.levenez.com/lang/  Versiones / Estándares  Prolog  1970  Haskell  1987  C#  2000  Java  1995  C++  1983  Smalltalk  1971  Scheme  1975  PL/I  1964  Simula  1964  Ada  1979  C  1971  Pascal  1970  CPL  1963  COBOL  1959  ALGOL  1958  FORTRAN  1954  Lisp  1958  Luis Hernández Yáñez  Fundamentos de la programación: Computadoras y programación Página 19  Luis Hernández Yáñez  La *prehistoria*  El ábaco  Siglo XIX  Máquina analítica de Charles Babbage  Fundamentos de la programación: Computadoras y programación Página 20  Lady Ada Lovelace  es considerada  la primera  programadora  (Wikipedia)  Luis Hernández Yáñez  Siglo XX  1936 Máquina de Turing  1946 ENIAC: Primera computadora digital  de propósito general  1947 El transistor  1953 IBM 650: Primera  computadora a gran escala  1966 ARPANET: Origen de Internet  1967 El *disquete*  1970 Sistema operativo UNIX  1972 Primer virus informático (*Creeper*)  Lenguaje de programación C  1974 Protocolo TCP. Primera red local  Fundamentos de la programación: Computadoras y programación Página 21  ENIAC (Wikipedia)  Luis Hernández Yáñez  1975 Se funda Microsoft  1976 Se funda Apple  1979 Juego *Pacman*  1981 IBM PC  Sistema operativo MS‐DOS  1983 Lenguaje de programación C++  1984 CD‐ROM  1985 Windows 1.0  1990 Lenguaje HTML  *World Wide Web*  1991 Sistema operativo Linux  Fundamentos de la programación: Computadoras y programación Página 22  Apple II (Wikipedia)  IBM PC (Wikipedia)  Linux  Luis Hernández Yáñez  1992 Windows 3.1  1995 Lenguaje de programación Java  DVD  1998 Se funda Google  1999 MSN Messenger  Siglo XXI  2001 Windows XP  Mac OS X  2002 Mozilla Firefox  2007 iPhone  2008 Android ...  Fundamentos de la programación: Computadoras y programación Página 23  Luis Hernández Yáñez  Fundamentos de la programación: Computadoras y programación Página 24  Luis Hernández Yáñez  *¿Qué es programar?*  *Decirle a un tonto* ***muy*** *rápido* ***exactamente*** *lo que tiene que hacer*  Especificar la estructura y el comportamiento de un programa,  así como probar que el programa realiza su tarea  adecuadamente y con un rendimiento aceptable  Programa: Transforma entrada en salida  Algoritmo: Secuencia de pasos y operaciones que debe realizar  el programa para resolver el problema  El programa implementa el algoritmo en un lenguaje concreto  Programa Salida  Fundamentos de la programación: Computadoras y programación Página 25  Entrada  Luis Hernández Yáñez  *La programación es sólo una etapa del proceso de desarrollo*  Modelo de desarrollo “en cascada”:  Fundamentos de la programación: Computadoras y programación Página 26  Mantenimiento  Prueba y depuración  Programación  Diseño  Análisis  Planificación Recursos necesarios, presupuesto, plan, …  ¿Qué?  ¿Cómo?  Implementación  Luis Hernández Yáñez  Fundamentos de la programación: Computadoras y programación Página 27  Luis Hernández Yáñez  *Bjarne Stroustrup (1983)*  #include <iostream>  using namespace std;  int main()  {  cout << "Hola Mundo!" << endl;  // Muestra Hola Mundo!  return 0;  }  Fundamentos de la programación: Computadoras y programación Página 28  Hola Mundo!  Luis Hernández Yáñez  Instrucciones  Datos: literales, variables, tipos  Subprogramas (funciones)  Comentarios  Directivas  ...  Fundamentos de la programación: Computadoras y programación Página 29  #include <iostream>  using namespace std;  int main()  {  cout << "Hola Mundo!" << endl;  // Muestra Hola Mundo!  return 0;  }  Directiva  Subprograma  Comentario  Instrucción  Instrucción  Dato  Dato  Luis Hernández Yáñez  Fundamentos de la programación: Computadoras y programación Página 30  Luis Hernández Yáñez  *Sintaxis y semántica de los lenguajes*  Sintaxis  — Reglas que determinan cómo se pueden construir  y secuenciar los elementos del lenguaje  Semántica  — Significado de cada elemento del lenguaje  ¿Para qué sirve?  Fundamentos de la programación: Computadoras y programación Página 31  Luis Hernández Yáñez  *Especificación*   Lenguajes (BNF)   Diagramas  Fundamentos de la programación: Computadoras y programación Página 32  <numero entero> ::= <signo opcional><secuencia de dígitos>  <signo opcional> ::= +|‐|<nada>  <secuencia de dígitos> ::= <dígito>|<dígito><secuencia de dígitos>  <dígito> ::= 0|1|2|3|4|5|6|7|8|9  <nada> ::=  BNF  | significa ó +23   ‐159   1374   1‐34   3.4   002   Ejemplo: Números enteros (sin decimales)  **+**  **0 .. 9**  **‐**  Luis Hernández Yáñez  Fundamentos de la programación: Computadoras y programación Página 33  <numero entero> ::= <signo opcional><secuencia de dígitos>  <signo opcional> ::= +|‐|<nada>  <secuencia de dígitos> ::= <dígito>|<dígito><secuencia de dígitos>  <dígito> ::= 0|1|2|3|4|5|6|7|8|9  <nada> ::=  **+23**  <numero entero> ::= <signo opcional><secuencia de dígitos>  ::= **+**<secuencia de dígitos> ::= **+**<dígito><secuencia de dígitos>  ::= **+2**<secuencia de dígitos> ::= **+2**<dígito> ::= **+23**  **1374**  <numero entero> ::= <signo opcional><secuencia de dígitos>  ::= <secuencia de dígitos> ::= <dígito><secuencia de dígitos>  ::= **1**<secuencia de dígitos> ::= **1**<dígito><secuencia de dígitos>  ::= **13**<secuencia de dígitos> ::= **13**<dígito><secuencia de dígitos>  ::= **137**<secuencia de dígitos> ::= **137**<dígito> ::= **1374**  **1‐34**  <numero entero> ::= <signo opcional><secuencia de dígitos>  ::= <secuencia de dígitos> ::= <dígito><secuencia de dígitos>  ::= **1**<secuencia de dígitos> ::= **ERROR** (**‐** no es <dígito>)        Luis Hernández Yáñez  Fundamentos de la programación: Computadoras y programación Página 34  +23   1374   ? 1‐   +23  1374  1‐34  **+**  **0 .. 9**  **+**  **+**  **‐**  **‐**  **‐**  **0 .. 9**  **0 .. 9**  Luis Hernández Yáñez  Fundamentos de la programación: Computadoras y programación Página 35  Luis Hernández Yáñez  *Hola Mundo!*  Un programa que muestra un saludo en la pantalla:  #include <iostream>  using namespace std;  int main()  // main() es donde empieza la ejecución  {  cout << "Hola Mundo!" << endl; // Muestra Hola Mundo!  return 0;  }  Fundamentos de la programación: Computadoras y programación Página 36  Luis Hernández Yáñez  *Análisis del programa*  #include <iostream>  using namespace std;  int main()  {  cout << "Hola Mundo!" << endl;  return 0;  }  Página 37  Las instrucciones terminan en ;  Fundamentos de la programación: Computadoras y programación  Directiva  Instrucción  Declaración  Instrucción  Instrucción  Biblioteca  Espacio de nombres  Tipo Palabras reservadas  Datos literales  Cuerpo de la función  Cabecera de la función  Bloque de código  Cadena de caracteres Constante  Operador Operador  Variable  Número  Coloreado sintáctico  Luis Hernández Yáñez  *Hola Mundo!*  Casi todo es *infraestructura*  Sólo  cout << "Hola Mundo!" << endl  hace algo palpable  La infraestructura (notación, bibliotecas y otro soporte)  hace nuestro código simple, completo, confiable y eficiente  *¡El estilo importa!*  Fundamentos de la programación: Computadoras y programación Página 38  Luis Hernández Yáñez  Fundamentos de la programación: Computadoras y programación Página 39  Luis Hernández Yáñez  *Editor*   Bloc de notas, Wordpad, Word, Writer, Gedit, Kwrite, …  (texto simple, sin formatos)   Editores específicos: coloreado sintáctico   Recomendación: Notepad++  Página 40  Instalación y uso:  Sección  Herramientas de desarrollo  en el Campus Virtual  Fundamentos de la programación: Computadoras y programación  Luis Hernández Yáñez  Fundamentos de la programación: Computadoras y programación Página 41  hola.cpp  (código fuente)  hola.obj  (código objeto)  Compilador  Código objeto de  la biblioteca iostream  hola.exe  (ejecutable)  Enlazador  Hola Mundo!  Cargador  Luis Hernández Yáñez  *Compilador*   Importante: C++ estándar   Recomendación: GNU G++ (*MinGW*en Windows)  Página 42  Instalación y uso:  Sección  Herramientas de desarrollo  en el Campus Virtual  Fundamentos de la programación: Computadoras y programación  Luis Hernández Yáñez  *Entornos de desarrollo*   Para editar, compilar y probar el código del programa   Recomendaciones:  — Windows: MS Visual Studio / C++ Express o Eclipse  — Linux: Netbeans o Eclipse  Página 43  Instalación y uso:  Sección  Herramientas de desarrollo  en el Campus Virtual  Fundamentos de la programación: Computadoras y programación  Luis Hernández Yáñez  *¿Qué hace el programa?*   La ejecución del programa siempre empieza en main()   Se ejecutan las instrucciones en secuencia de principio a fin  Página 44  \_  Hola Mundo!  Pantalla (cout)  \_  Muestra Hola Mundo!  en la pantalla y salta de línea  Devuelve 0 como código  de terminación del programa  Fin  return 0;  cout << "Hola Mundo!" << endl;  Inicio  Fundamentos de la programación: Computadoras y programación  Luis Hernández Yáñez  Fundamentos de la programación: Computadoras y programación Página 45  Luis Hernández Yáñez  *El lenguaje C*   Lenguaje creado por Dennis M. Ritchie en 1972   Lenguaje de nivel medio:  — Estructuras típicas de los lenguajes de alto nivel  — Construcciones para control a nivel de máquina   Lenguaje sencillo (pocas palabras reservadas)   Lenguaje estructurado (no estrictamente estructurado en bloques)   Compartimentalización de código (funciones) y datos (ámbitos)   Componente estructural básico: la función (subprograma)   Programación modular   Distingue entre mayúsculas y minúsculas   Palabras reservadas (o clave): en minúsculas  Fundamentos de la programación: Computadoras y programación Página 46  Luis Hernández Yáñez  *Licencia CC (Creative Commons)*  Este tipo de licencias ofrecen algunos derechos a terceras personas  bajo ciertas condiciones.  Este documento tiene establecidas las siguientes:  Pulsa en la imagen de arriba a la derecha para saber más.  Fundamentos de la programación: Computadoras y programación Página 47  Reconocimiento (*Attribution*):  En cualquier explotación de la obra autorizada por la licencia  hará falta reconocer la autoría.  No comercial (*Non commercial*):  La explotación de la obra queda limitada a usos no comerciales.  Compartir igual (*Share alike*):  La explotación autorizada incluye la creación de obras derivadas  siempre que mantengan la misma licencia al ser divulgadas.  **2**  Grado en Ingeniería Informática  Grado en Ingeniería del Software  Grado en Ingeniería de Computadores  Luis Hernández Yáñez  Facultad de Informática  Universidad Complutense  Fundamentos de la programación  Luis Hernández Yáñez  Un ejemplo de programación 50  El primer programa en C++ 64  Las líneas de código del programa 80  Cálculos en los programas 86  Variables 92  Expresiones 98  Lectura de datos desde el teclado 108  Resolución de problemas 119  Los datos de los programas 127  Identificadores 129  Tipos de datos 133  Declaración y uso de variables 142  Instrucciones de asignación 147  Operadores 152  Más sobre expresiones 160  Constantes 167  La biblioteca cmath 171  Operaciones con caracteres 174  Operadores relacionales 177  Toma de decisiones (if) 180  Bloques de código 183  Bucles (while) 186  Entrada/salida por consola 190  Funciones definidas  por el programador 199  Fundamentos de la programación: Tipos e instrucciones I  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 50  Luis Hernández Yáñez  *Una computadora de un coche*  Instrucciones que entiende:  <instrucción> ::= <inst> ;  <inst> ::= Start | Stop | <avanzar>  <avanzar> ::= Go <dirección> <num> Blocks  <dirección> ::= North | East | South | West  <num> ::= 1 | 2 | 3 | 4 | 5  Ejemplos:  Start;  Go North 3 Blocks;  Stop;  Fundamentos de la programación: Tipos e instrucciones I Página 51  Luis Hernández Yáñez  *Sintaxis del lenguaje de programación*  Fundamentos de la programación: Tipos e instrucciones I Página 52  avanzar Go dirección num Blocks  = Literales  dirección  North  East  South  West  instrucción  Start  avanzar  Stop ;  num  1  2  3  4  5  Luis Hernández Yáñez  *El problema a resolver*  *Estando el coche en la posición A,*  *conseguir llegar al Cine Tívoli (B)*  ¿Qué pasos hay que seguir?  *Arrancar*  *Ir un bloque al Norte*  *Ir dos bloques al Este*  *Ir cinco bloques al Norte*  *Ir dos bloques al Este*  *Parar*  Fundamentos de la programación: Tipos e instrucciones I Página 53  B  A  **N**  Bloque:  Luis Hernández Yáñez  *El algoritmo*  Secuencia de pasos que hay que  seguir para resolver el problema  Fundamentos de la programación: Tipos e instrucciones I Página 54  *1.‐ Arrancar*  **N**  *2.‐ Ir un bloque al Norte*  *3.‐ Ir dos bloques al Este*  *4.‐ Ir cinco bloques al Norte*  *5.‐ Ir dos bloques al Este*  *6.‐ Parar*  Esos pasos sirven tanto para  una persona como para una computadora.    Luis Hernández Yáñez  *El programa*  Instrucciones escritas en  el lenguaje de programación  Start;  Go North 1 Blocks;  Go East 2 Blocks;  Go North 5 Blocks;  Go East 2 Blocks;  Stop;  Fundamentos de la programación: Tipos e instrucciones I Página 55  B  A  **N**  Luis Hernández Yáñez  *El programa*  Escribimos el código del programa en un editor  y lo guardamos en un archivo:  Fundamentos de la programación: Tipos e instrucciones I Página 56  Stat;  Go North 1 Blocks  Go East Blocks;  Go Noth 5 Blocks;  Go West 2 Blocks;  Stop;  Copiamos el archivo  en una llave USB  y lo llevamos al coche  Luis Hernández Yáñez  *La compilación*  Introducimos la llave USB en el coche  y pulsamos el botón de ejecutar el programa:  Fundamentos de la programación: Tipos e instrucciones I Página 57  Stat;  ‐‐‐‐^ Unknown word.  Go North 1 Blocks  ‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐^ ; missing.  Go East Blocks;  ‐‐‐‐‐‐‐‐^ Number missing.  Go Noth 5 Blocks;  ‐‐‐‐‐‐‐^ Unknown word.  Go West 2 Blocks;  Stop;  There are errors. Impossible to run the program.  Errores  de sintaxis  Luis Hernández Yáñez  *Depuración*  Editamos el código para corregir los errores sintácticos:  Fundamentos de la programación: Tipos e instrucciones I Página 58  Stat;  Go North 1 Blocks  Go East Blocks;  Go Noth 5 Blocks;  Go West 2 Blocks;  Stop;  Start;  Go North 1 Blocks;  Go East 3 Blocks;  Go North 5 Blocks;  Go West 2 Blocks;  Stop;  Luis Hernández Yáñez  *La ejecución*  Se realiza lo que pide  cada instrucción:  Fundamentos de la programación: Tipos e instrucciones I Página 59  B  **N**  Start;  Go North 1 Blocks;  Go East 3 Blocks; !  Error de ejecución  *¡Una instrucción no se puede ejecutar!*  Luis Hernández Yáñez  *Depuración*  Editamos el código para arreglar el error de ejecución:  Fundamentos de la programación: Tipos e instrucciones I Página 60  Start;  Go North 1 Blocks;  Go East 2 Blocks;  Go North 5 Blocks;  Go West 2 Blocks;  Stop;  Start;  Go North 1 Blocks;  Go East 3 Blocks;  Go North 5 Blocks;  Go West 2 Blocks;  Stop;  Luis Hernández Yáñez  *La ejecución*  Se realiza lo que pide  cada instrucción:  Fundamentos de la programación: Tipos e instrucciones I Página 61  B  **N**  Start;  Go North 1 Blocks;  Go East 2 Blocks;  Error lógico  *¡El programa no llega al resultado deseado!*  Go North 5 Blocks;  Go West 2 Blocks;  Stop;  ?  Luis Hernández Yáñez  *Depuración*  Editamos el código para arreglar el error lógico:  Fundamentos de la programación: Tipos e instrucciones I Página 62  Start;  Go North 1 Blocks;  Go East 2 Blocks;  Go North 5 Blocks;  Go West 2 Blocks;  Stop;  Start;  Go North 1 Blocks;  Go East 2 Blocks;  Go North 5 Blocks;  Go East 2 Blocks;  Stop;  Luis Hernández Yáñez  *La ejecución*  Se realiza lo que pide  cada instrucción:  Fundamentos de la programación: Tipos e instrucciones I Página 63  **N**  Start;  Go North 1 Blocks;  Go East 2 Blocks;  Go North 5 Blocks;  Go East 2 Blocks;  Stop;    *¡Conseguido!*  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 64  Luis Hernández Yáñez  *Hola Mundo!*  De vuelta en el programa que muestra un saludo en la pantalla:  #include <iostream>  using namespace std;  int main() // main() es donde empieza la ejecución  {  cout << "Hola Mundo!" << endl;  return 0;  }  Fundamentos de la programación: Tipos e instrucciones I Página 65  Luis Hernández Yáñez  *Hola Mundo!*  La única instrucción que produce algo tangible:  #include <iostream>  using namespace std;  int main() // main() es donde empieza la ejecución  {  cout << "Hola Mundo!" << endl;  return 0;  }  Fundamentos de la programación: Tipos e instrucciones I Página 66  Luis Hernández Yáñez  cout (iostream)  Visualización en la pantalla: operador << (*insertor*)  Fundamentos de la programación: Tipos e instrucciones I Página 67  cout << "Hola Mundo!" << endl;  cout  \_  Hola Mundo!  << "Hola Mundo!" << endl;  endl  *end line*  c*haracter* out*put stream*  Luis Hernández Yáñez  *Pantalla en modo texto*   Líneas de 80 caracteres (textos)  Fundamentos de la programación: Tipos e instrucciones I Página 68  Aplicación en modo texto  80 caracteres  Luis Hernández Yáñez  *Ventanas de consola o terminal*  Las aplicaciones en modo texto se ejecutan dentro de ventanas:   Windows: ventanas de consola (*Símbolo del sistema*)   Linux: ventanas de terminal  Fundamentos de la programación: Tipos e instrucciones I Página 69  **H o l a M u n d o !**  Cursor parpadeante: Donde se colocará el siguiente carácter.  ...  ...  Luis Hernández Yáñez  *El insertor* <<  *Inserta textos en la pantalla de modo texto*  Representación textual de los datos  A partir de la posición del cursor  *Line wrap* (continúa en la siguiente línea si no cabe)  Se pueden encadenar:  cout << ... << ... << ...;  Fundamentos de la programación: Tipos e instrucciones I Página 70  cout << ...;  Recuerda: las instrucciones terminan en ;  Luis Hernández Yáñez  *Con el insertor* << *podemos mostrar...*   Cadenas de caracteres literales  Textos encerrados entre comillas dobles: "..."  cout << "Hola Mundo!";   Números literales  Con o sin decimales, con signo o no: 123, ‐37, 3.1416, ...  cout << "Pi = " << 3.1416;  Se muestran los caracteres que representan el número   endl  Fundamentos de la programación: Tipos e instrucciones I Página 71  *¡Las comillas no se muestran!*  *¡Punto decimal, NO coma!*  Luis Hernández Yáñez  *El programa principal*  La función main(): *donde comienza la ejecución...*  #include <iostream>  using namespace std;  int main() // main() es donde empieza la ejecución  {  cout << "Hola Mundo!" << endl;  return 0;  }  Contiene las instrucciones que hay que ejecutar  Fundamentos de la programación: Tipos e instrucciones I Página 72  Luis Hernández Yáñez  *El programa principal*  La función main():  int main()  {  ...  return 0;  }  Fundamentos de la programación: Tipos e instrucciones I Página 73  Tipo de la función (int = entero): Tipo de valor que devuelve  Nombre de la función  Cuerpo de la función (bloque de código)  return 0; Devuelve el resultado (0) de la función  *¡Es una función!*  Luis Hernández Yáñez  *Documentando el código...*  Comentarios (se ignoran):  #include <iostream>  using namespace std;  int main() // main() es donde empieza la ejecución  {  cout << "Hola Mundo!" << endl;  ...  Hasta el final de la línea: // Comentario de una línea  De varias líneas: /\* Comentario de varias  líneas seguidas \*/  Fundamentos de la programación: Tipos e instrucciones I Página 74  Luis Hernández Yáñez  *La infraestructura*  Código para reutilizar:  #include <iostream>  using namespace std;  int main() // main() es donde empieza la ejecución  {  cout << "Hola Mundo!" << endl;  return 0;  }  Bibliotecas de funciones a nuestra disposición  Fundamentos de la programación: Tipos e instrucciones I Página 75  Una directiva: empieza por #  Luis Hernández Yáñez  *Bibliotecas*  Se incluyen con la *directiva* #include:  #include <iostream>  (Utilidades de entrada/salida por consola)  Para mostrar o leer datos hay que incluir la biblioteca iostream  *Espacios de nombres*  En iostream hay espacios de nombres; ¿cuál queremos?  #include <iostream>  using namespace std;  Siempre usaremos el espacio de nombres estándar (std)  Muchas bibliotecas no tienen espacios de nombres  Fundamentos de la programación: Tipos e instrucciones I Página 76  Es una instrucción: termina en ;  Luis Hernández Yáñez  *Compilación y enlace*  Página 77  hola.cpp  (código fuente)  hola.obj  (código objeto)  Compilador  Código objeto de  la biblioteca iostream  hola.exe  (ejecutable)  Enlazador  Fundamentos de la programación: Tipos e instrucciones I  Hola Mundo!  Cargador  A menudo en un paso  Luis Hernández Yáñez  *Elementos del programa*  #include <iostream>  using namespace std;  int main()  {  cout << "Hola Mundo!" << endl;  return 0;  }  Página 78  Las instrucciones terminan en ;  Fundamentos de la programación: Tipos e instrucciones I  Directiva  Instrucción  Declaración  Instrucción  Instrucción  Biblioteca  Espacio de nombres  Tipo Palabras reservadas  Datos literales  Cuerpo de la función  Cabecera de la función  Bloque de código  Cadena de caracteres Constante  Operador Operador  Variable  Número  Coloreado sintáctico:  Directivas Tipos  Palabras reservadas generales  Datos literales Comentarios  Luis Hernández Yáñez  *Uso de espacio en blanco*  Separación de elementos por uno o más *espacios en blanco*  (espacios, tabuladores y saltos de línea)  El compilador los ignora  Página 79  #include <iostream>  using namespace std;  int main()  {  cout << "Hola Mundo!" << endl;  return 0;  }  #include <iostream> using namespace std;  int main(){cout<<"Hola Mundo!"<<endl;  return 0;}  ¿Cuál se lee mejor?  Fundamentos de la programación: Tipos e instrucciones I  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 80  Luis Hernández Yáñez  *Programa con E/S por consola*  Una plantilla para empezar:  #include <iostream>  using namespace std;  int main()  {  return 0;  }  Fundamentos de la programación: Tipos e instrucciones I Página 81  *¡Tu código aquí!*  Luis Hernández Yáñez  *... recitado en la consola*  Mostrar los textos con cout <<:  #include <iostream>  using namespace std;  int main()  {  cout << "En un lugar de la Mancha," << endl;  cout << "de cuyo nombre no quiero acordarme," << endl;  cout << "no ha mucho tiempo que vivía un hidalgo de los de  lanza en astillero, ..." << endl;  return 0;  }  Fundamentos de la programación: Tipos e instrucciones I Página 82  Luis Hernández Yáñez  *Introducción del código del programa*  Terminamos cada línea de código con un salto de línea (↲):  #include <iostream> ↲  using namespace std; ↲  ↲  int main() ↲  { ↲  cout << "En un lugar de la Mancha," << endl; ↲  cout << "de cuyo nombre no quiero acordarme," << endl; ↲  cout << "no ha mucho tiempo que vivía un hidalgo de los de  lanza en astillero, ..." << endl; ↲  return 0; ↲  } ↲  Fundamentos de la programación: Tipos e instrucciones I Página 83  Luis Hernández Yáñez  *Introducción del código del programa*  No hay que partir una cadena literal entre dos líneas:  cout << "no ha mucho tiempo que vivía un hidalgo de ↲  los de lanza en astillero, ..." << endl; ↲  *¡La cadena no termina (1ª línea)!*  *¡No se entiende* los *(2ª línea)!*  *Veamos cómo nos muestra los errores el compilador...*  Fundamentos de la programación: Tipos e instrucciones I Página 84  Luis Hernández Yáñez  *Mantenimiento y reusabilidad*   Usa espacio en blanco para separar los elementos:  cout << "En un lugar de la Mancha," << endl;  mejor que  cout<<"En un lugar de la Mancha,"<<endl;   Usa sangría (indentación) para el código de un bloque:  {  cout << "En un lugar de la Mancha," << endl;  ...  return 0;  }  *¡El estilo importa!*  Fundamentos de la programación: Tipos e instrucciones I Página 85  Tab  ó  3 esp.  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 86  Luis Hernández Yáñez  *Operadores aritméticos*  + Suma  ‐ Resta  \* Multiplicación  / División  Operadores binarios  *operando\_izquierdo operador operando\_derecho*  Operación Resultado  3 + 4 7  2.56 ‐ 3 ‐0.44  143 \* 2 286  45.45 / 3 15.15  Fundamentos de la programación: Tipos e instrucciones I Página 87  Luis Hernández Yáñez  *Números literales (concretos)*   Enteros: sin parte decimal  Signo negativo (opcional) + secuencia de dígitos  3 143 ‐12 67321 ‐1234   Reales: con parte decimal  Signo negativo (opcional) + secuencia de dígitos  + punto decimal + secuencia de dígitos  3.1416 357.0 ‐1.333 2345.6789 ‐404.1  Fundamentos de la programación: Tipos e instrucciones I Página 88  Punto decimal (3.1416), NO coma (3,1416)  No se usan puntos de millares  Luis Hernández Yáñez  *Ejemplo*  #include <iostream>  using namespace std;  int main()  {  cout << "133 + 1234 = " << 133 + 1234 << endl;  cout << "1234 ‐ 111.5 = " << 1234 ‐ 111.5 << endl;  cout << "34 \* 59 = " << 34 \* 59 << endl;  cout << "3.4 \* 5.93 = " << 3.4 \* 5.93 << endl;  cout << "500 / 3 = " << 500 / 3 << endl; // Div. entera  cout << "500.0 / 3 = " << 500.0 / 3 << endl; // Div. real  return 0;  }  Fundamentos de la programación: Tipos e instrucciones I Página 89  cálculos.cpp  Un texto Un número  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 90  División real  División entera  Luis Hernández Yáñez  *¿División entera o división real?*  Ambos operandos enteros  División entera  Algún operando real  División real  División Resultado  500 / 3 166  500.0 / 3 166.667  500 / 3.0 166.667  500.0 / 3.0 166.667  Fundamentos de la programación: Tipos e instrucciones I Página 91  Comprueba siempre que el tipo de división sea el que quieres  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 92  Luis Hernández Yáñez  *Datos que se mantienen en memoria*  Variable: dato que se accede por medio de un nombre  Dato literal: un valor concreto  Variable: puede cambiar de valor (*variar*)  edad = 19; // variable edad y literal 19  Las variables deben ser declaradas  ¿Qué tipo de dato queremos mantener?   Valor numérico sin decimales (entero): tipo int   Valor numérico con decimales (real): tipo double  Declaración: *tipo nombre*;  Fundamentos de la programación: Tipos e instrucciones I Página 93  Luis Hernández Yáñez  *Declaración de variables*  int cantidad;  double precio;  Se reserva espacio suficiente  LAS VARIABLES NO SE INICIALIZAN  No se deben usar hasta que se les haya dado algún valor  *¿Dónde colocamos las declaraciones?*  Siempre, antes del primer uso  Habitualmente al principio de la función  Fundamentos de la programación: Tipos e instrucciones I Página 94  Memoria  cantidad **?**  precio **?**  **...**  *tipo nombre*;  Luis Hernández Yáñez  *Declaración de variables*  #include <iostream>  using namespace std;  int main()  {  int cantidad;  double precio, total;  return 0;  }  Fundamentos de la programación: Tipos e instrucciones I Página 95  Memoria  cantidad **?**  precio **?**  total **?**  **...**  Podemos declarar varias de un mismo tipo  separando los nombres con comas  Luis Hernández Yáñez  *Capacidad de las variables*  int  ‐2.147.483.648 ... 2.147.483.647  ‐2147483648 .. 2147483647  double  2,23 x 10‐308 ... 1,79 x 10+308 y sus negativos  [+|‐] 2.23e‐308 .. 1.79e+308  Problemas de precisión  Fundamentos de la programación: Tipos e instrucciones I Página 96  Notación científica  Luis Hernández Yáñez  *Asignación de valores a las variables (operador* =*)*  *variable* = *expresión*;  cantidad = 12; // int  precio = 39.95; // double  total = cantidad \* precio; // Asigna 479.4  Concordancia de tipos: cantidad = 12.5;  *¡¡¡A la izquierda del = debe ir siempre una variable!!!*  Fundamentos de la programación: Tipos e instrucciones I Página 97  Instrucción: termina en ;  cantidad  12  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 98  Luis Hernández Yáñez  *Expresiones*  Secuencias de operandos y operadores  *operando operador operando operador operando ...*  total = cantidad \* precio \* 1.18;  A igual prioridad se evalúan de izquierda a derecha  Paréntesis para forzar ciertas operaciones  total = cantidad1 + cantidad2 \* precio;  total = (cantidad1 + cantidad2) \* precio;  Unos operadores se evalúan antes que otros  Fundamentos de la programación: Tipos e instrucciones I Página 99  Expresión    Luis Hernández Yáñez  *Precedencia de los operadores*  cantidad1 = 10;  cantidad2 = 2;  precio = 40.0;  \* y / se evalúan antes que + y ‐  total = cantidad1 + cantidad2 \* precio;  \* antes que +  10 + 2 \* 40,0  10 + 80,0  90,0  total = (cantidad1 + cantidad2) \* precio;  + antes que \*  (10 + 2) \* 40,0  12 \* 40,0  480,0  Fundamentos de la programación: Tipos e instrucciones I Página 100  Luis Hernández Yáñez  *Ejemplo de uso de variables y expresiones*  #include <iostream>  using namespace std;  int main()  {  int cantidad;  double precio, total;  cantidad = 12;  precio = 39.95;  total = cantidad \* precio;  cout << cantidad << " x " << precio << " = "  << total << endl;  return 0;  }  Fundamentos de la programación: Tipos e instrucciones I Página 101  variables.cpp  Luis Hernández Yáñez  Memoria  **...**  *Ejemplo de uso de variables*  #include <iostream>  using namespace std;  int main()  {  int cantidad;  double precio, total;  Fundamentos de la programación: Tipos e instrucciones I Página 102  cantidad **?**  precio **?**  total **?**  Luis Hernández Yáñez  *Ejemplo de uso de variables*  #include <iostream>  using namespace std;  int main()  {  int cantidad;  double precio, total;  cantidad = 12;  Fundamentos de la programación: Tipos e instrucciones I Página 103  Memoria  **...**  cantidad **12**  precio **?**  total **?**  Luis Hernández Yáñez  *Ejemplo de uso de variables*  #include <iostream>  using namespace std;  int main()  {  int cantidad;  double precio, total;  cantidad = 12;  precio = 39.95;  Fundamentos de la programación: Tipos e instrucciones I Página 104  Memoria  **...**  cantidad **12**  precio **39.95**  total **?**  Luis Hernández Yáñez  *Ejemplo de uso de variables*  #include <iostream>  using namespace std;  int main()  {  int cantidad;  double precio, total;  cantidad = 12;  precio = 39.95;  total = cantidad \* precio;  Fundamentos de la programación: Tipos e instrucciones I Página 105  Memoria  **...**  cantidad **12**  precio **39.95**  total **479.4**  Luis Hernández Yáñez  *Ejemplo de uso de variables*  #include <iostream>  using namespace std;  int main()  {  int cantidad;  double precio, total;  cantidad = 12;  precio = 39.95;  total = cantidad \* precio;  cout << cantidad << " x " << precio << " = "  << total << endl;  Fundamentos de la programación: Tipos e instrucciones I Página 106  Memoria  **...**  cantidad **12**  precio **39.95**  total **479.4**  Luis Hernández Yáñez  *Ejemplo de uso de variables*  #include <iostream>  using namespace std;  int main()  {  int cantidad;  double precio, total;  cantidad = 12;  precio = 39.95;  total = cantidad \* precio;  cout << cantidad << " x " << precio << " = "  << total << endl;  return 0;  }  Fundamentos de la programación: Tipos e instrucciones I Página 107  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 108  Luis Hernández Yáñez  cin (iostream)  Lectura de valores de variables: operador >> (*extractor*)  Fundamentos de la programación: Tipos e instrucciones I Página 109  cin >> cantidad;  **cin** >> cantidad;  c*haracter* in*put stream*  Memoria  cantidad **?**  **...**  \_  **12**  1 2 ↲  1 2  Luis Hernández Yáñez  *El extractor* >>  *Transforma los caracteres introducidos en datos*  Cursor parpadeante: lugar de lectura del siguiente carácter  La entrada termina con Intro (cursor a la siguiente línea)  *¡El destino del extractor debe ser SIEMPRE una variable!*  Se ignoran los espacios en blanco iniciales  Fundamentos de la programación: Tipos e instrucciones I Página 110  cin >> *variable*;  Luis Hernández Yáñez  *Lectura de valores enteros (*int*)*  Se leen dígitos hasta encontrar un carácter que no lo sea  12abc↲ 12 abc↲ 12 abc↲ 12↲  Se asigna el valor 12 a la variable  El resto queda pendiente para la siguiente lectura  Recomendación: Lee cada variable en una línea 12↲  *Lectura de valores reales (*double*)*  Se leen dígitos, el punto decimal y otros dígitos  39.95.5abc↲ 39.95 abc↲ 39.95↲  Se asigna el valor 39,95 a la variable; el resto queda pendiente  Recomendación: Lee cada variable en una línea 39.95↲  Fundamentos de la programación: Tipos e instrucciones I Página 111  Luis Hernández Yáñez  *¿Qué pasa si el usuario se equivoca?*  El dato no será correcto  Aplicación profesional: código de comprobación y ayuda  Aquí supondremos que los usuarios no se equivocan  En ocasiones añadiremos comprobaciones sencillas  Fundamentos de la programación: Tipos e instrucciones I Página 112  Para evitar errores, lee cada dato en una instrucción aparte  Luis Hernández Yáñez  *¿Qué pasa si el usuario se equivoca?*  int cantidad;  double precio, total;  cout << "Introduce la cantidad: ";  cin >> cantidad;  cout << "Introduce el precio: ";  cin >> precio;  cout << "Cantidad: " << cantidad << endl;  cout << "Precio: " << precio << endl;  Fundamentos de la programación: Tipos e instrucciones I Página 113  No se puede leer un entero  0 para cantidad y Error  La lectura del precio falla: precio no toma valor (*basura*)  *¡Amigable con el usuario!*  ¿Qué tiene que introducir?  Luis Hernández Yáñez  *¿Qué pasa si el usuario se equivoca?*  Fundamentos de la programación: Tipos e instrucciones I Página 114  12 para cantidad  No se puede leer un real   0 para precio y Error  12 para cantidad  .5  0,5 para precio  Lo demás queda pendiente  *¡¡¡Lectura correcta!!!*  Luis Hernández Yáñez  *División de dos números*  *Pedir al usuario dos números y mostrarle el resultado*  *de dividir el primero entre el segundo*  Algoritmo.‐  1. Pedir el numerador  2. Pedir el denominador  3. Realizar la división, guardando el resultado  4. Mostrar el resultado  Fundamentos de la programación: Tipos e instrucciones I Página 115  Datos / cálculos  Variable denominador (double)  Variable resultado (double)  resultado = numerador / denominador  Variable numerador (double)  Luis Hernández Yáñez  Declaraciones Entrada Procesamiento Salida  *Entrada‐Proceso‐Salida*  Muchos programas se ajustan a un sencillo esquema:  Fundamentos de la programación: Tipos e instrucciones I Página 116  1. Leer numerador  2. Leer denominador  3. Calcular división en resultado  4. Mostrar resultado  Luis Hernández Yáñez  *División de dos números*  *Pedir al usuario dos números y mostrarle el resultado de dividir el*  *primero entre el segundo.*  1. Leer numerador  2. Leer denominador  3. Calcular división en resultado  4. Mostrar resultado  Fundamentos de la programación: Tipos e instrucciones I Página 117  cin >> numerador;  cin >> denominador;  resultado = numerador / denominador;  cout << resultado;  Luis Hernández Yáñez  *División de dos números*  #include <iostream>  using namespace std;  int main()  {  double numerador, denominador, resultado;  cout << "Numerador: ";  cin >> numerador;  cout << "Denominador: ";  cin >> denominador;  resultado = numerador / denominador;  cout << "Resultado: " << resultado << endl;  return 0;  }  Salida  Procesamiento  Entrada  Declaraciones  Fundamentos de la programación: Tipos e instrucciones I Página 118  división.cpp  \_  R\_esultado: \_64.5  Numerador: \_  Denominador: \_  129  2  \_  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 119  Luis Hernández Yáñez  *Problema*  *Dadas la base y la altura de un triángulo, mostrar su área*  *Mostrar en la pantalla un texto que pida la base del triángulo. El usuario*  *introducirá el valor con el teclado. Mostrar en la pantalla un texto que*  *pida la altura del triángulo. El usuario introducirá el valor con el teclado.*  *Se calculará el área del triángulo y se mostrará en la pantalla.*  Fundamentos de la programación: Tipos e instrucciones I Página 120  Refinamiento  Luis Hernández Yáñez  *Objetos: Datos que maneja el programa*  *Mostrar en la pantalla un texto que pida la base del triángulo. El usuario*  *introducirá la base con el teclado. Mostrar en la pantalla un texto que*  *pida la altura del triángulo. El usuario introducirá la altura con el*  *teclado. Se calculará el área del triángulo y se mostrará en la pantalla.*  Fundamentos de la programación: Tipos e instrucciones I Página 121  cout cadena literal  variable cin  cadena literal variable  variable  Luis Hernández Yáñez  *Datos que maneja el programa: tipos*  Fundamentos de la programación: Tipos e instrucciones I Página 122  *Objeto Tipo ¿Varía? Nombre*  Pantalla Variable cout  "Introduzca la base del triángulo: " Constante *ninguno*  Base del triángulo double Variable base  Teclado Variable cin  "Introduzca la altura del triángulo: " Constante *ninguno*  Altura del triángulo double Variable altura  Área del triángulo double Variable area  Luis Hernández Yáñez  *Operaciones (acciones)*  *Mostrar en la pantalla un texto que pida la base del triángulo. El usuario*  *introducirá la base con el teclado. Mostrar en la pantalla un texto que*  *pida la altura del triángulo. El usuario introducirá la altura con el*  *teclado. Se calculará el área del triángulo y se mostrará en la pantalla.*  Fundamentos de la programación: Tipos e instrucciones I Página 123  cout << ... cin >> ...  area = base \* altura / 2  Luis Hernández Yáñez  Secuencia de acciones que ha de realizar el programa  para conseguir resolver el problema  1. Mostrar en la pantalla el texto que pida la base del triángulo  2. Leer del teclado el valor para la base del triángulo  3. Mostrar en la pantalla el texto que pida la altura  4. Leer del teclado el valor para la altura del triángulo  5. Calcular el área del triángulo  6. Mostrar el área del triángulo  Fundamentos de la programación: Tipos e instrucciones I Página 124  Luis Hernández Yáñez  #include <iostream>  using namespace std;  int main()  {  return 0;  }  Fundamentos de la programación: Tipos e instrucciones I Página 125  Algoritmo  traducido  a código  en C++  Declaraciones  1. Mostrar en la pantalla el texto que pida la base del triángulo  2. Leer del teclado el valor para la base del triángulo  3. Mostrar en la pantalla el texto que pida la altura del triángulo  4. Leer del teclado el valor para la altura del triángulo  5. Calcular el área del triángulo  6. Mostrar el área del triángulo  Luis Hernández Yáñez  *El programa: implementación*  #include <iostream>  using namespace std;  int main()  {  double base, altura, area; // Declaraciones  cout << "Introduzca la base del triángulo: "; // 1  cin >> base; // 2  cout << "Introduzca la altura del triángulo: "; // 3  cin >> altura; // 4  area = base \* altura / 2; // 5  cout << "El área de un triángulo de base " << base // 6  << " y altura " << altura << " es: " << area << endl;  return 0;  }  Fundamentos de la programación: Tipos e instrucciones I Página 126  triángulo.cpp  Recuerda: las instrucciones terminan en ;  ¿triβngulo?  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 127  Luis Hernández Yáñez  *Variabilidad de los datos*  Fundamentos de la programación: Tipos e instrucciones I Página 128  Datos  Constantes  Literales  Con nombre  Variables  "Introduzca la base del triángulo: "  3.141592653589  base, altura, area  Pi = 3.141592653589  Identificadores  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 129  Luis Hernández Yáñez  Para variables y constantes con nombre  — *Nombre* de un dato (para accederlo/modificarlo)  — Deben ser descriptivos  Sintaxis:  Fundamentos de la programación: Tipos e instrucciones I Página 130  Al menos 32 caracteres significativos  0..9, a..z, A..Z, \_  a..z, A..Z, \_  *¡Ni eñes ni vocales acentuadas!*   palabras reservadas  cantidad prrecio total base altura area numerador  Luis Hernández Yáñez  *Palabras reservadas del lenguaje C++*  asm auto bool break case catch char class const  const\_cast continue default delete do double  dynamic\_cast else enum explicit extern false  float for friend goto if inline int long  mutable namespace new operator private protected  public register reinterpret\_cast return short  signed sizeof static static\_cast struct switch  template this throw true try typedef typeid  typename union unsigned using virtual void  volatile while  Fundamentos de la programación: Tipos e instrucciones I Página 131  Luis Hernández Yáñez  *¿Qué identificadores son válidos?*  balance interesAnual  \_base\_imponible años  EDAD12 salario\_1\_mes  \_\_edad cálculoNómina  valor%100 AlgunValor  100caracteres valor?  \_12\_meses \_\_\_\_valor  Fundamentos de la programación: Tipos e instrucciones I Página 132                          0..9, a..z, A..Z, \_  a..z, A..Z, \_  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 133  Luis Hernández Yáñez  *Tipos*  Cada dato, de un tipo concreto  Cada tipo establece:  — El conjunto (intervalo) de valores válidos  — El conjunto de operaciones que se pueden realizar  Expresiones con datos de distintos tipos (compatibles):  Transformación automática de tipos (*promoción de tipo*)  Fundamentos de la programación: Tipos e instrucciones I Página 134  125  true 'a'  3.14159  "Hola"  *Anexo del Tema 2: detalles técnicos*  Luis Hernández Yáñez  int  Números enteros (sin decimales) 1363, ‐12, 49  float  Números reales 12.45, ‐3.1932, 1.16E+02  double  Números reales (mayores intervalo y precisión)  char  Caracteres 'a' , '{', '\t'  bool  Valores lógicos (verdadero/falso) true, false  string  Cadenas de caracteres (biblioteca string) "Hola Mundo!"  void  *Nada*, ausencia de tipo, ausencia de dato (*funciones*)  Fundamentos de la programación: Tipos e instrucciones I Página 135      Luis Hernández Yáñez  Intervalo de valores: Juego de caracteres (ASCII)  Literales:  'a' '%' '\t'  Constantes de barra invertida (o *secuencias de escape*):  Caracteres de control  '\t' = tabulador '\n' = salto de línea …  Fundamentos de la programación: Tipos e instrucciones I Página 136  ASCII (códigos 32..127)  ISO‐8859‐1  (ASCII extendido: códigos 128..255)  1 byte  *Caracteres*  Luis Hernández Yáñez  Sólo dos valores posibles:  — Verdadero (*true*)  — Falso (*false*)  Literales:  true false  Cualquier número distinto de 0 es equivalente a true  El 0 es equivalente a false  Fundamentos de la programación: Tipos e instrucciones I Página 137  *Valores lógicos*  Luis Hernández Yáñez  *C++ distingue entre mayúsculas y minúsculas*  int: palabra reservada de C++ para declarar datos enteros  Int, INT o inT no son palabras reservadas de C++  true: palabra reservada de C++ para el valor *verdadero*  True o TRUE no son palabras reservadas de C++  Fundamentos de la programación: Tipos e instrucciones I Página 138  Luis Hernández Yáñez  "Hola" "Introduce el numerador: " "X142FG5TX?%A"  Secuencias de caracteres  Programas con variables de tipo string:  #include <string>  using namespace std;  Fundamentos de la programación: Tipos e instrucciones I Página 139  " "  Las comillas tipográficas (apertura/cierre) “…” NO sirven  Asegúrate de utilizar comillas rectas: "…"  char  *Cadenas de caracteres*  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 140  tipos.cpp  #include <iostream>  #include <string>  using namespace std; // Un solo using... para ambas bibliotecas  int main()  {  int entero = 3; // Podemos asignar (inicializar) al declarar  double real = 2.153;  char caracter = 'a';  bool cierto = true;  string cadena = "Hola";  cout << "Entero: " << entero << endl;  cout << "Real: " << real << endl;  cout << "Carácter: " << caracter << endl;  cout << "Booleano: " << cierto << endl;  cout << "Cadena: " << cadena << endl;  return 0;  }  *¿Cuántos números hay en total en el programa?*  *¿Y caracteres? ¿Y cadenas? ¿Y booleanos?*  Luis Hernández Yáñez  — signed / unsigned : con signo (por defecto) / sin signo  — short / long : menor / mayor intervalo de valores  Fundamentos de la programación: Tipos e instrucciones I Página 141  Tipo Intervalo  int ‐2147483648 .. 2147483647  unsigned int 0 .. 4294967295  short int ‐32768 .. 32768  unsigned short int 0 .. 65535  long int ‐2147483648 .. 2147483647  unsigned long int 0 .. 4294967295  double +|‐ 2.23e‐308 .. 1.79e+308  long double +|‐ 3.37E‐4932 .. 1.18E+4932  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 142  Luis Hernández Yáñez  [modificadores] tipo lista\_de\_variables;  Opcional  lista\_de\_variables  int i, j, l;  short int unidades;  unsigned short int monedas;  double balance, beneficio, perdida;  Fundamentos de la programación: Tipos e instrucciones I Página 143  *Identificador*  *Programación con buen estilo*:  Identificadores descriptivos  Espacio tras cada coma  Nombres de las variables en minúsculas  (Varias palabras: capitaliza cada inicial: interesPorMes)  ,  Luis Hernández Yáñez  Se reserva memoria suficiente para cada tipo de dato  int inicio;  short int unidades;  double balance;  Fundamentos de la programación: Tipos e instrucciones I Página 144  Memoria  01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  **. . .**  inicio  unidades  balance  Luis Hernández Yáñez  *¡En C++ las variables no se inicializan automáticamente!*  *¡Una variable debe ser haber sido inicializada antes de ser accedida!*  ¿Cómo se inicializa una variable?  — Al leer su valor (cin >>)  — Al asignarle un valor (instrucción de asignación)  — Al declararla  Inicialización en la propia declaración:  Fundamentos de la programación: Tipos e instrucciones I Página 145  … *Identificador* = *Expresión* Expresión: valor compatible  En particular, una expresión  puede ser un literal  int i = 0, j, l = 26;  short int unidades = 100;  Luis Hernández Yáñez  *Obtención del valor de una variable*   Nombre de la variable en una expresión  cout << balance;  cout << interesPorMes \* meses / 100;  *Modificación del valor de una variable*   Nombre de la variable a la izquierda del =  balance = 1214;  porcentaje = valor / 30;  Las variables han de haber sido previamente declaradas  Fundamentos de la programación: Tipos e instrucciones I Página 146  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 147  Luis Hernández Yáñez  *El operador =*  A la izquierda, SIEMPRE una variable  int i, j = 2;  i = 23 + j \* 5; // i toma el valor 33  = *Expresión* ;  Fundamentos de la programación: Tipos e instrucciones I Página 148  *Variable*  Luis Hernández Yáñez  *Errores*  int a, b, c;  5 = a;  // ERROR: un literal no puede recibir un valor  a + 23 = 5;  // ERROR: no puede haber una expresión a la izda.  b = "abc";  // ERROR: un entero no puede guardar una cadena  c = 23 5;  // ERROR: expresión no válida (falta operador)  Fundamentos de la programación: Tipos e instrucciones I Página 149  Luis Hernández Yáñez  int i, j = 2;  i = 23 + j \* 5;  Fundamentos de la programación: Tipos e instrucciones I Página 150  Memoria  01  02  03  04  05  06  07  08  09  10  **. . .**  i  j 2  ?  Memoria  01  02  03  04  05  06  07  08  09  10  **. . .**  i  j 2  23 + 2 \* 5 33  Luis Hernández Yáñez  Necesitamos una variable auxiliar  double a = 3.45, b = 127.5, aux;  Fundamentos de la programación: Tipos e instrucciones I Página 151  a 3.45  b 127.5  aux ?  a 3.45  b 127.5  aux 3.45  a 127.5  b 127.5  aux 3.45  a 127.5  b 3.45  aux 3.45  aux = a;  a = b;  b = aux;  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 152  Luis Hernández Yáñez  *Operaciones sobre valores de los tipos*  Cada tipo determina las operaciones posibles  Tipos de datos numéricos (int, float y double):  — Asignación (=)  — Operadores aritméticos  — Operadores relacionales (menor, mayor, igual, ...)  Tipo de datos bool:  — Asignación (=)  — Operadores lógicos (Y, O, NO)  Tipos de datos char y string:  — Asignación (=)  — Operadores relacionales (menor, mayor, igual, ...)  Fundamentos de la programación: Tipos e instrucciones I Página 153  Luis Hernández Yáñez  *Operadores para tipos de datos numéricos*  Fundamentos de la programación: Tipos e instrucciones I Página 154  Operador Operandos Posición int float / double  ‐ 1 (monario) Prefijo Cambio de signo  + 2 (binario) Infijo Suma  ‐ 2 (binario) Infijo Resta  \* 2 (binario) Infijo Producto  / 2 (binario) Infijo Div. entera División real  % 2 (binario) Infijo Módulo No aplicable  ++ 1 (monario) Prefijo / postfijo Incremento  ‐‐ 1 (monario) Prefijo / postfijo Decremento  Luis Hernández Yáñez  *Operadores monarios y operadores binarios*  Operadores monarios (*unarios*)  — Cambio de signo (‐):  Delante de variable, constante o expresión entre paréntesis  ‐saldo ‐RATIO ‐(3 \* a ‐ b)  — Incremento/decremento (sólo variables) (prefijo/postfijo):  ++interes ‐‐meses j++ // 1 más ó 1 menos  Operadores binarios  — Operando izquierdo operador operando derecho  Operandos: literales, constantes, variables o expresiones  2 + 3 a \* RATIO ‐a + b  (a % b) \* (c / d)  Fundamentos de la programación: Tipos e instrucciones I Página 155  Luis Hernández Yáñez  *¿División entera o división real?*  Ambos operandos enteros: división entera  int i = 23, j = 2;  cout << i / j; // Muestra 11  Algún operando real: división real  int i = 23;  double j = 2;  cout << i / j; // Muestra 11.5  Fundamentos de la programación: Tipos e instrucciones I Página 156  /  Luis Hernández Yáñez  *Módulo (resto de la división entera)*  Ambos operandos han de ser enteros  int i = 123, j = 5;  cout << i % j; // Muestra 3  División entera:  No se obtienen decimales  Queda un resto  Fundamentos de la programación: Tipos e instrucciones I Página 157  %  123 5  3 24  123 % 5  Luis Hernández Yáñez  *Operadores de incremento y decremento*  Incremento/decremento de la variable numérica en una unidad  Prefijo: Antes de acceder  int i = 10, j;  j = ++i; // Incrementa antes de copiar  cout << i << " ‐ " << j; // Muestra 11 ‐ 11  Postfijo: Después de acceder  int i = 10, j;  j = i++; // Copia y después incrementa  cout << i << " ‐ " << j; // Muestra 11 ‐ 10  Fundamentos de la programación: Tipos e instrucciones I Página 158  ++/‐‐  No mezcles ++ y ‐‐ con otros operadores  i=i+1;  j=i;  j=i;  i=i+1;  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 159  #include <iostream> operadores.cpp  using namespace std;  int main() {  int entero1 = 15, entero2 = 4;  double real1 = 15.0, real2 = 4.0;  cout << "Operaciones entre los números 15 y 4:" << endl;  cout << "División entera (/): " << entero1 / entero2 << endl;  cout << "Resto de la división (%): " << entero1 % entero2 << endl;  cout << "División real (/): " << real1 / real2 << endl;  cout << "Num = " << real1 << endl;  real1 = ‐real1;  cout << "Cambia de signo (‐): " << real1 << endl;  real1 = ‐real1;  cout << "Vuelve a cambiar (‐): " << real1 << endl;  cout << "Se incrementa antes (++ prefijo): " << ++real1 << endl;  cout << "Se muestra antes de incrementar (posfijo ++): "  << real1++ << endl;  cout << "Ya incrementado: " << real1 << endl;  return 0;  }  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 160  Luis Hernández Yáñez  *¿En qué orden se evalúan los operadores?*  3 + 5 \* 2 / 2 ‐ 1  ¿De izquierda a derecha?  ¿De derecha a izquierda?  ¿Unos antes que otros?  Precedencia de los operadores (prioridad):  Se evalúan antes los de mayor precedencia  ¿Y si tienen igual prioridad?  Normalmente, de izquierda a derecha  Paréntesis: fuerzan a evaluar su subexpresión  Fundamentos de la programación: Tipos e instrucciones I Página 161  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 162  Precedencia Operadores  Mayor prioridad ++ ‐‐ (postfijos)  ++ ‐‐ (prefijos)  ‐ (cambio de signo)  \* / %  Menor prioridad + ‐  3 + 5 \* 2 / 2 ‐ 1  Misma precedencia:  Izquierda antes  Misma precedencia:  Izquierda antes  Mayor  precedencia  3 + 10 / 2 ‐ 1 3 + 5 ‐ 1  8 ‐ 1  7  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 163  ((3 + 5) \* 4 + 12) / 4 ‐ (3 \* 2 ‐ 1) Primero, los paréntesis...  \* antes que ‐  (8 \* 4 + 12) / 4 ‐ (6 ‐ 1)  \* antes que +  (32 + 12) / 4 ‐ 5  44 / 4 ‐ 5  11 ‐ 5  6  / antes que ‐  Pon espacio antes y después  de cada operador binario  Luis Hernández Yáñez  #include <iostream>  using namespace std;  int main()  {  double x, f;  cout << "Introduce el valor de X: ";  cin >> x;  f = 3 \* x \* x / 5 + 6 \* x / 7 ‐ 3;  cout << "f(x) = " << f << endl;  return 0;  }  Fundamentos de la programación: Tipos e instrucciones I Página 164  fórmula.cpp  Usa paréntesis para mejorar la legibilidad:  f = (3 \* x \* x / 5) + (6 \* x / 7) ‐ 3;  Luis Hernández Yáñez  *variable* = *variable operador op\_derecho*;  *variable operador*= *op\_derecho*;  *Asignación Abreviatura*  a = a + 12; a += 12;  a = a \* 3; a \*= 3;  a = a ‐ 5; a ‐= 5;  a = a / 37; a /= 37;  a = a % b; a %= b;  Fundamentos de la programación: Tipos e instrucciones I Página 165  Igual precedencia  que la asignación  De momento,  mejor evitarlas  La misma   Luis Hernández Yáñez  *¿Valor siguiente al máximo?*  Valor mayor del máximo (o menor del mínimo) del tipo  short int i = 32767; // Valor máximo para short int  i++; // 32768 no cabe en un short int  cout << i; // Muestra ‐32768  Bit de signo  0 = positivo  1 = negativo  Fundamentos de la programación: Tipos e instrucciones I Página 166  **0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1**  **1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**  + **0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1**  32767  ‐32768  + 1  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 167  Luis Hernández Yáñez  *Declaración de constantes* Modificador de acceso const  *Variables inicializadas a las que no dejamos variar*  Fundamentos de la programación: Tipos e instrucciones I Página 168  *Declaración de variable con inicializador*  *Programación con buen estilo*:  Pon en mayúscula la primera letra  de una constante o todo su nombre  La constante no podrá volver a  aparecer a la izquierda de un =  const  const short int Meses = 12;  const double Pi = 3.141592,  RATIO = 2.179 \* Pi;  Luis Hernández Yáñez   Aumentan la legibilidad del código  cambioPoblacion = (0.1758 ‐ 0.1257) \* poblacion; *vs.*  cambioPoblacion = (RatioNacimientos ‐ RatioMuertes) \* poblacion;   Facilitan la modificación del código  double compra1 = bruto1 \* 18 / 100;  double compra2 = bruto2 \* 18 / 100;  double total = compra1 + compra2;  cout << total << " (IVA: " << 18 << "%)" << endl;  const int IVA = 18;  double compra1 = bruto1 \* IVA / 100;  double compra2 = bruto2 \* IVA / 100;  double total = compra1 + compra2;  cout << total << " (IVA: " << IVA << "%)" << endl;  Fundamentos de la programación: Tipos e instrucciones I Página 169  ¿Cambio del IVA al 21%?  3 cambios  1 cambio  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 170  constantes.cpp  #include <iostream>  using namespace std;  int main() {  const double Pi = 3.141592;  double radio = 12.2, circunferencia;  circunferencia = 2 \* Pi \* radio;  cout << "Circunferencia de un círculo de radio "  << radio << ": " << circunferencia << endl;  const double Euler = 2.718281828459; // Número e  cout << "Número e al cuadrado: " << Euler \* Euler << endl;  const int IVA = 21;  int cantidad = 12;  double precio = 39.95, neto, porIVA, total;  neto = cantidad \* precio;  porIVA = neto \* IVA / 100;  total = neto + porIVA;  cout << "Total compra: " << total << endl;  return 0;  }  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 171  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 172  abs(x) Valor absoluto de x  pow(x, y) x elevado a y  sqrt(x) Raíz cuadrada de x  ceil(x) Menor entero que es mayor o igual que x  floor(x) Mayor entero que es menor o igual que x  exp(x) ex  log(x) Ln x (logaritmo natural de x)  log10(x) Logaritmo en base 10 de x  sin(x) Seno de x  cos(x) Coseno de x  tan(x) Tangente de x  round(x) Redondeo al entero más próximo  trunc(x) Pérdida de la parte decimal (entero)  #include <cmath>  Algunas ...  Luis Hernández Yáñez  #include <iostream>  using namespace std;  #include <cmath>  int main() {  double x, y, f;  cout << "Valor de X: ";  cin >> x;  cout << "Valor de Y: ";  cin >> y;  f = 2 \* pow(x, 5) + sqrt(pow(x, 3) / pow(y, 2))  / abs(x \* y) ‐ cos(y);  cout << "f(x, y) = " << f << endl;  return 0;  }  Fundamentos de la programación: Tipos e instrucciones I Página 173  Pon un espacio detrás de cada coma en las listas de argumentos  mates.cpp  pow() con argumento entero:  Usa el molde double():  pow(double(i), 5)  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 174  Luis Hernández Yáñez  Asignación, ++/‐‐ y operadores relacionales  *Funciones para caracteres* (biblioteca cctype)  isalnum(c) true si c es una letra o un dígito  isalpha(c) true si c es una letra  isdigit(c) true si c es un dígito  islower(c) true si c es una letra minúscula  isupper(c) true si c es una letra mayúscula  toupper(c) devuelve la mayúscula de c  tolower(c) devuelve la minúscula de c  …  Fundamentos de la programación: Tipos e instrucciones I Página 175  char  false en caso contrario  Luis Hernández Yáñez  ...  #include <cctype>  int main() {  char caracter1 = 'A', caracter2 = '1', caracter3 = '&';  cout << "Carácter 1 (" << caracter1 << ").‐" << endl;  cout << "Alfanumérico? " << isalnum(caracter1) << endl;  cout << "Alfabético? " << isalpha(caracter1) << endl;  cout << "Dígito? " << isdigit(caracter1) << endl;  cout << "Mayúscula? " << isupper(caracter1) << endl;  caracter1 = tolower(caracter1);  cout << "En minúscula: " << caracter1 << endl;  cout << "Carácter 2 (" << caracter2 << ").‐" << endl;  cout << "Alfabético? " << isalpha(caracter2) << endl;  cout << "Dígito? " << isdigit(caracter2) << endl;  cout << "Carácter 3 (" << caracter3 << ").‐" << endl;  cout << "Alfanumérico? " << isalnum(caracter3) << endl;  cout << "Alfabético? " << isalpha(caracter3) << endl;  cout << "Dígito? " << isdigit(caracter3) << endl;  return 0;  }  Fundamentos de la programación: Tipos e instrucciones I Página 176  caracteres.cpp  1  true / 0  false  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 177  Luis Hernández Yáñez  *Operadores relacionales*  Comparaciones (*condiciones*)  Condición simple ::= Expresión Operador\_relacional Expresión  Concordancia de tipo entre las expresiones  Resultado: bool (true o false)  Fundamentos de la programación: Tipos e instrucciones I Página 178  < menor que  <= menor o igual que  > mayor que  >= mayor o igual que  == igual que  != distinto de  Operadores (prioridad)  ...  \* / %  + ‐  < <= > >=  == !=  = += ‐= \*= /= %=  Luis Hernández Yáñez  Menor prioridad que los operadores aditivos y multiplicativos  bool resultado;  int a = 2, b = 3, c = 4;  resultado = a < 5; // 2 < 5  true  resultado = a \* b + c >= 12; // 10 >= 12  false  resultado = a \* (b + c) >= 12; // 14 >= 12  true  resultado = a != b; // 2 != 3  true  resultado = a \* b > c + 5; // 6 > 9  false  resultado = a + b == c + 1; // 5 == 5  true  Fundamentos de la programación: Tipos e instrucciones I Página 179  No confundas el operador de igualdad (==)  con el operador de asignación (=)  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 180  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 181  *Selección: bifurcación condicional*  true  *códigoT*  false  *códigoF*  if (*condición*) {  *códigoT*  }  else {  *códigoF*  }  *Condición*  int num;  cout << "Número: ";  cin >> num;  if (num % 2 == 0) {  cout << num << " es par";  }  else {  cout << num << " es impar";  }  Opcional: puede no haber else  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 182  #include <iostream>  using namespace std;  int main() {  int op1 = 13, op2 = 4;  int opcion;  cout << "1 ‐ Sumar" << endl;  cout << "2 ‐ Restar" << endl;  cout << "Opción: ";  cin >> opcion;  if (opcion == 1) {  cout << op1 + op2 << endl;  }  else {  cout << op1 ‐ op2 << endl;  }  return 0;  }  selección.cpp  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 183  Luis Hernández Yáñez  {  *intrucción1*  *intrucción2*  ...  *intrucciónN*  }  Fundamentos de la programación: Tipos e instrucciones I Página 184  *Agrupación de instrucciones*  Grupo de instrucciones a ejecutar en una rama del if  Tab ó  3 esp.  int num, total = 0;  cin >> num;  if (num > 0)  {  cout << "Positivo";  total = total + num;  }  cout << endl;  Ámbito local  (declaraciones locales)  { *instrucción* }  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 185  *Posición de las llaves: cuestión de estilo*  if (num > 0) if (num > 0) {  { cout << "Positivo";  cout << "Positivo"; total = total + num;  total = total + num; }  } cout << endl;  cout << endl;  *No necesitamos las llaves si sólo hay una instrucción*  if (num > 0) { if (num > 0)  cout << "Positivo"; cout << "Positivo";  }  Usaremos siempre llaves por simplicidad...  Evita poner el if y la instrucción objetivo en la misma línea:  if (num > 0) cout << "Positivo";    Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 186  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 187  *Repetición o iteración condicional*  while (*condición*) {  *cuerpo*  }  *Cuerpo*  true false  Bloque  de código  Si la condición es false al empezar,  no se ejecuta el cuerpo ninguna vez  while ( *condición* ) *cuerpo*  *Condición*  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 188  #include <iostream>  using namespace std;  int main() {  int i = 1, n = 0, suma = 0;  while (n <= 0) { // Sólo n positivo  cout << "¿Cuántos números quieres sumar? ";  cin >> n;  }  while (i <= n) {  suma = suma + i;  i++;  }  cout << "Sumatorio de i (1 a " << n << ") = "  << suma << endl;  return 0;  }  serie.cpp  Luis Hernández Yáñez  false  *Iteración condicional*  Fundamentos de la programación: Tipos e instrucciones I Página 189  while (i <= n) {  suma = suma + i;  i++;  }  n i 1  suma 0  suma += i;  i++;  true  2  1  3  3  4  6  5  10  5  6  15  i <= n  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 190  Luis Hernández Yáñez  **7 3 5 . 3 5 1 1 6 = l a t o T**  *Flujos de texto (streams)*  Conectan la ejecución del programa con los dispositivos de E/S  Son secuencias de caracteres  Entrada por teclado: flujo de entrada cin (tipo istream)  Salida por pantalla: flujo de salida cout (tipo ostream)  Fundamentos de la programación: Tipos e instrucciones I Página 191  Programa  cin cout  Biblioteca iostream con espacio de nombres std  *Flujo de entrada* >> *Variable*  *Flujo de salida* << *Expresión*  Extractor  Insertor  #include <iostream>  using namespace std;  Luis Hernández Yáñez  Salta los *espacios en blanco* (espacios, tabuladores o saltos de línea)  — char  Se lee un carácter en la variable  — int  Se leen dígitos y se transforman en el valor a asignar  — float/double:  Se leen dígitos (quizá el punto y más dígitos) y se asigna el valor  — bool:  Si se lee 1, se asigna true; con cualquier otro valor se asigna false  Fundamentos de la programación: Tipos e instrucciones I Página 192  cin >> *Variable*  Se amigable con el usuario  Lee cada dato en una línea  cout << "Introduce tu edad: ";  cin >> edad;  Luis Hernández Yáñez  cin >> *cadena* termina con el primer espacio en blanco  cin.sync() descarta la entrada pendiente  Fundamentos de la programación: Tipos e instrucciones I Página 193  *¿Cómo leer varias palabras?*  *Siguiente página...*  apellidos recibe "Antonio"  string nombre, apellidos;  cout << "Nombre: ";  cin >> nombre;  cout << "Apellidos: ";  cin >> apellidos;  cout << "Nombre completo: "  << nombre << " "  << apellidos << endl;  string nombre, apellidos;  cout << "Nombre: ";  cin >> nombre;  cin.sync();  cout << "Apellidos: ";  cin >> apellidos;  cout << ...  #include <string>  using namespace std;  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 194  *Lectura sin saltar los espacios en blanco iniciales*  Llamada a funciones con el operador punto (.) :  El operador punto permite llamar a una función sobre una variable  *variable*.*función*(*argumentos*)  Lectura de un carácter sin saltar espacios en blanco:  cin.get(c); // Lee el siguiente carácter  Lectura de cadenas sin saltar los espacios en blanco:  getline(cin, *cad*);  Lee todo lo que haya hasta el final de la línea (Intro)  Recuerda:  *Espacios en blanco* son espacios, tabuladores, saltos de línea, ...  Luis Hernández Yáñez  *Representación textual de los datos*  int meses = 7;  cout << "Total: " << 123.45 << endl << " Meses: " << meses;  El valor double 123.45 se guarda en memoria en binario  Su representación textual es:'1' '2' '3' '.' '4' '5'  Fundamentos de la programación: Tipos e instrucciones I Página 195  *Expresión*  La biblioteca iostream  define la constante endl  como un salto de línea  cout <<  d 123.45  **5 4 . 3 2 1**  *¡Un número real!*  *¡Un texto!*  *(secuencia de caracteres)*  double d = 123.45;  cout << d;  Luis Hernández Yáñez  int meses = 7;  cout << "Total: " << 123.45 << endl << " Meses: " << meses;  Fundamentos de la programación: Tipos e instrucciones I Página 196  **T o t a l : 1 2 3 . 4 5**  **M e s e s : 7** Programa  cout  cout << 123.45 << endl << " Meses: " << meses;  cout << endl << " Meses: " << meses;  cout << " Meses: " << meses;  cout << meses; Total: 123.45  Meses: 7  cout << *Expresión*  Luis Hernández Yáñez  Constantes y funciones a enviar a cout para ajustar el formato de salida  Fundamentos de la programación: Tipos e instrucciones I Página 197  Biblioteca Constante/función Propósito  iostream showpoint /  noshowpoint  Mostrar o no el punto decimal para reales sin  decimales (34.0)  fixed Notación de punto fijo (reales) (123.5)  scientific Notación científica (reales) (1.235E+2)  boolalpha Valores bool como true / false  left / right Ajustar a la izquierda/derecha (por defecto)  iomanip setw(*anchura*)\* Nº de caracteres (anchura) para el dato  setprecision(*p*) Precisión: Nº de dígitos (en total)  Con fixed o scientific, nº de decimales  \*setw() sólo afecta al siguiente dato que se escriba,  mientras que los otros afectan a todos  #include <iomanip>  Luis Hernández Yáñez  bool fin = false;  cout << fin << "‐>" << boolalpha << fin << endl;  double d = 123.45;  char c = 'x';  int i = 62;  cout << d << c << i << endl;  cout << "|" << setw(8) << d << "|" << endl;  cout << "|" << left << setw(8) << d << "|" << endl;  cout << "|" << setw(4) << c << "|" << endl;  cout << "|" << right << setw(5) << i << "|" << endl;  double e = 96;  cout << e << " ‐ " << showpoint << e << endl;  cout << scientific << d << endl;  cout << fixed << setprecision(8) << d << endl;  Fundamentos de la programación: Tipos e instrucciones I Página 198  0‐>false  123.45x62  | 123.45|  |123.45 |  |x |  | 62|  96 ‐ 96.0000  1.234500e+002  123.45000000  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones I Página 199  Luis Hernández Yáñez  Los programas pueden incluir otras funciones además de main()  Forma general de una función en C++:  *tipo nombre*(*parámetros*) // Cabecera  {  // *Cuerpo*  }   *Tipo* de dato que devuelve la función como resultado   *Parámetros* para proporcionar datos a la función  Declaraciones de variables separadas por comas   *Cuerpo*: secuencia de declaraciones e instrucciones  ¡Un bloque de código!  Fundamentos de programación: Tipos e instrucciones I Página 200  Luis Hernández Yáñez   Datos locales: declarados en el cuerpo de la función  Datos auxiliares que utiliza la función (puede no haber)   Parámetros: declarados en la cabecera de la función  Datos de entrada de la función (puede no haber)  Ambos son de uso exclusivo de la función y no se conocen fuera  double f(int x, int y) {  // Declaración de datos locales:  double resultado;  // Instrucciones:  resultado = 2 \* pow(x, 5) + sqrt(pow(x, 3)  / pow(y, 2)) / abs(x \* y) ‐ cos(y);  return resultado; // Devolución del resultado  }  Fundamentos de programación: Tipos e instrucciones I Página 201  Luis Hernández Yáñez  *Llamada a una función con parámetros*  *Nombre*(*Argumentos*)  Al llamar a la función:  — Tantos argumentos entre los paréntesis como parámetros  — Orden de declaración de los parámetros  — Cada argumento: mismo tipo que su parámetro  — Cada argumento: expresión válida (se pasa el resultado)  Se copian los valores resultantes de las expresiones  en los correspondientes parámetros  Llamadas a la función: en expresiones de otras funciones  int valor = f(2, 3);  Fundamentos de programación: Tipos e instrucciones I Página 202  Luis Hernández Yáñez  *Se copian los argumentos en los parámetros*  int funcion(int x, double a) {  ...  }  int main() {  int i = 124;  double d = 3;  funcion(i, 33 \* d);  ...  return 0; // main() devuelve 0 al S.O.  }  Fundamentos de programación: Tipos e instrucciones I Página 203  Memoria  i 124  d 3.0  **...**  **...**  x 124  a 99.0  **...**  Los argumentos no se modifican  Luis Hernández Yáñez  *La función ha de devolver un resultado*  La función termina su ejecución devolviendo un resultado  La instrucción return *(sólo una en cada función)*  — Devuelve el dato que se pone a continuación (tipo de la función)  — Termina la ejecución de la función  El dato devuelto sustituye a la llamada de la función:  int cuad(int x) { int main() {  return x \* x; cout << 2 \* cuad(16);  x = x \* x;  } return 0;  }  Fundamentos de programación: Tipos e instrucciones I Página 204  256  Esta instrucción  no se ejecutará nunca  Luis Hernández Yáñez  *¿Qué funciones hay en el programa?*  Colocaremos las funciones después de main()  *¿Son correctas las llamadas a funciones del programa?*  — ¿Existe la función?  — ¿Concuerdan los argumentos con los parámetros?   Prototipos tras las inclusiones de bibliotecas  Prototipo de función: Cabecera de la función terminada en ;  double f(int x, int y);  int funcion(int x, double a)  int cuad(int x);  ...  Fundamentos de programación: Tipos e instrucciones I Página 205  main() es la única función  que no hay que prototipar  Luis Hernández Yáñez  #include <iostream>  using namespace std;  #include <cmath>  // Prototipos de las funciones (excepto main())  bool par(int num);  bool letra(char car);  int suma(int num);  double formula(int x, int y);  int main() {  int numero, sum, x, y;  char caracter;  double f;  cout << "Entero: ";  cin >> numero;  if (par(numero)) {  cout << "Par";  }  ...  Fundamentos de programación: Tipos e instrucciones I Página 206  Luis Hernández Yáñez  else {  cout << "Impar";  }  cout << endl;  if (numero > 1) {  cout << "Sumatorio de 1 a " << numero << ": "  << suma(numero) << endl;  }  cout << "Carácter: ";  cin >> caracter;  if (!letra(caracter)) {  cout << "no ";  }  cout << "es una letra" << endl;  cout << "f(x, y) = " << formula(x, y) << endl;  // Los argumentos pueden llamarse igual o no que los parámetros  return 0;  }  ...  Fundamentos de programación: Tipos e instrucciones I Página 207  Luis Hernández Yáñez  // Implementación de las funciones propias  bool par(int num) {  bool esPar;  if (num % 2 == 0) {  esPar = true;  }  else {  esPar = false;  }  return esPar;  }  ...  Fundamentos de programación: Tipos e instrucciones I Página 208  Luis Hernández Yáñez  bool letra(char car) {  bool esLetra;  if ((car >= 'a') && (car <= 'z') || (car >= 'A') && (car <= 'Z')) {  esLetra = true;  }  else {  esLetra = false;  }  return esLetra;  }  int suma(int num) {  int sum = 0, i = 1;  while (i < num) {  sum = sum + i;  i++;  }  return sum;  }  ...  Fundamentos de programación: Tipos e instrucciones I Página 209  Luis Hernández Yáñez  double formula(int x, int y) {  double f;  f = 2 \* pow(x, 5) + sqrt(pow(x, 3) / pow(y, 2))  / abs(x \* y) ‐ cos(y);  return f;  }  Fundamentos de programación: Tipos e instrucciones I Página 210  funciones.cpp  Luis Hernández Yáñez  *Licencia CC (Creative Commons)*  Este tipo de licencias ofrecen algunos derechos a terceras personas  bajo ciertas condiciones.  Este documento tiene establecidas las siguientes:  Pulsa en la imagen de arriba a la derecha para saber más.  Fundamentos de la programación: Tipos e instrucciones I Página 211  Reconocimiento (*Attribution*):  En cualquier explotación de la obra autorizada por la licencia  hará falta reconocer la autoría.  No comercial (*Non commercial*):  La explotación de la obra queda limitada a usos no comerciales.  Compartir igual (*Share alike*):  La explotación autorizada incluye la creación de obras derivadas  siempre que mantengan la misma licencia al ser divulgadas.  **2A**  Grado en Ingeniería Informática  Grado en Ingeniería del Software  Grado en Ingeniería de Computadores  Luis Hernández Yáñez  Facultad de Informática  Universidad Complutense  Fundamentos de la programación  Luis Hernández Yáñez  int 214  float 216  Notación científica 217  double 218  char 220  bool 221  string 222  Literales con especificación de tipo 223  Fundamentos de la programación: Tipos e instrucciones I (Anexo)  Luis Hernández Yáñez  Intervalo de valores:  ‐2147483648 .. 2147483647  Bytes de memoria: 4\*  Literales:  1363, ‐12, 010 , 0x1A  Fundamentos de la programación: Tipos e instrucciones I (Anexo) Página 214  01  02  03  04  05  06  07  08  09  **. . .**  Notación hexadecimal  (\*) Depende de la máquina  4 bytes es lo más habitual  Se puede saber cuántos  se usan con la función  sizeof(int)  Notación octal  +  ‐  0  x 0..9,A..F  0..9  0..9  0..7  *Números enteros*  Luis Hernández Yáñez  Números en notación octal (base 8: dígitos entre 0 y 7):  ‐010 = ‐8 en notación decimal  10 = 1 x 81 + 0 x 80 = 1 x 8 + 0  0423 = 275 en notación decimal  423 = 4 x 82 + 2 x 81 + 3 x 80 = 4 x 64 + 2 x 8 + 3 = 256  + 16 +3  Números en notación hexadecimal (base 16):  *Dígitos* posibles: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F  0x1F = 31 en notación decimal  1F = 1 x 161 + F x 160 = 1 x 16 + 15  0xAD = 173 en notación decimal  AD = A x 161 + D x 160 = 10 x 16 + 13 = 160 + 13  Fundamentos de la programación: Tipos e instrucciones I (Anexo) Página 215  *Números enteros*  Luis Hernández Yáñez  Intervalo de valores:  +/‐ 1.18e‐38 .. 3.40e+38  Bytes de memoria: 4\*  Punto flotante. Precisión: 7 dígitos  Literales (punto decimal):   Notación normal: 134.45, ‐1.1764   Notación científica: 1.4E2, ‐5.23e‐02  Fundamentos de la programación: Tipos e instrucciones I (Anexo) Página 216  01  02  03  04  05  06  07  08  09  **. . .**  (\*)sizeof(float)  0..9  +  ‐  +  ‐  +  ‐  .  .  e,E  0..9  0..9 0..9 0..9  *Números reales (con decimales)*  Luis Hernández Yáñez  Siempre un número (con o sin signo) con un solo dígito de parte  entera, seguido del exponente (potencia de 10):  ‐5.23e‐2  ‐5,23 x 10‐2  ‐0,0523  1.11e2  1,11 x 102  111,0  7.4523e‐04  7,4523 x 10‐4  0,00074523  ‐3.3333e+06  ‐3,3333 x 106  ‐3.333.300  Fundamentos de la programación: Tipos e instrucciones I (Anexo) Página 217  Luis Hernández Yáñez  Intervalo de valores:  +/‐ 2.23e‐308 .. 1.79e+308  Bytes de memoria: 8\*  Punto flotante. Precisión: 15 dígitos  Literales (punto decimal):   Notación normal: 134.45, ‐1.1764   Notación científica: 1.4E2, ‐5.23e‐02  Fundamentos de la programación: Tipos e instrucciones I (Anexo) Página 218  01  02  03  04  05  06  07  08  09  **. . .**  (\*)sizeof(double)  0..9  +  ‐  ‐  +  ‐  .  .  e,E  0..9  0..9 0..9 0..9  +  *Números reales (con decimales)*  Luis Hernández Yáñez  Intervalo de valores:  Juego de caracteres (ASCII)  Bytes de memoria: 1 (FC)  Literales:  'a', '%', '\t'  Constantes de barra invertida:  (O *secuencias de escape*)  Para caracteres de control  '\t' = tabulador, '\n' = salto de línea, …  Fundamentos de la programación: Tipos e instrucciones I (Anexo) Página 219  01  02  03  04  05  06  07  08  09  **. . .**  *Carácter*  n, t, v, b, r, f, a, \  ' '  \  *Caracteres*  Luis Hernández Yáñez  Juego de caracteres ASCII:  ***A****merican* ***S****tandard* ***C****ode for* ***I****nformation* ***I****nterchange* (1963)  Caracteres con códigos entre 0 y 127 (7 bits)  — Caracteres de control:  Códigos del 0 al 31 y 127  Tabulación, salto de línea,...  — Caracteres imprimibles:  Códigos del 32 al 126  Juego de caracteres ASCII extendido (8 bits):  ISO‐8859‐1  + Códigos entre 128 y 255  Fundamentos de la programación: Tipos e instrucciones I (Anexo) Página 220  Multitud de codificaciones:  EBCDIC, UNICODE, UTF‐8, ...  Luis Hernández Yáñez  Sólo dos valores posibles:  — Verdadero (*true*)  — Falso (*false*)  Bytes de memoria: 1 (FC)  Literales:  true, false  En realidad, cualquier número  distinto de 0 es equivalente a true  y el número 0 es equivalente a false  Fundamentos de la programación: Tipos e instrucciones I (Anexo) Página 221  01  02  03  04  05  06  07  08  09  **. . .**  *Valores lógicos*  Luis Hernández Yáñez  "Hola", "Introduce el numerador: ", "X142FG5TX?%A"  Secuencias de caracteres  Se asigna la memoria que se necesita para la secuencia concreta  Requieren la biblioteca string con el espacio de nombres std:  #include <string>  using namespace std;  Fundamentos de la programación: Tipos e instrucciones I (Anexo) Página 222  *Carácter*  *¡Ojo!*  Las comillas tipográficas (apertura/cierre) “…” te darán problemas  al compilar. Asegúrate de utilizar comillas rectas: "…"  " "  *Cadenas de caracteres*  Luis Hernández Yáñez  Por defecto un literal entero se considera un dato int  — long int: 35L, 1546l  — unsigned int: 35U, 1546u  — unsigned long int: 35UL, 1546ul  Por defecto un literal real se considera un dato double  — float: 1.35F, 15.46f  — long double: 1.35L, 15.46l  *Abreviaturas para modificadores de tipos*  short  short int  long  long int  Es preferible evitar el uso de tales abreviaturas:  *Minimizar la cantidad de información a recordar*  *sobre el lenguaje*  Fundamentos de la programación: Tipos e instrucciones I (Anexo) Página 223  Luis Hernández Yáñez  *Licencia CC (Creative Commons)*  Este tipo de licencias ofrecen algunos derechos a terceras personas  bajo ciertas condiciones.  Este documento tiene establecidas las siguientes:  Pulsa en la imagen de arriba a la derecha para saber más.  Fundamentos de la programación: Tipos e instrucciones I (Anexo) Página 224  Reconocimiento (*Attribution*):  En cualquier explotación de la obra autorizada por la licencia  hará falta reconocer la autoría.  No comercial (*Non commercial*):  La explotación de la obra queda limitada a usos no comerciales.  Compartir igual (*Share alike*):  La explotación autorizada incluye la creación de obras derivadas  siempre que mantengan la misma licencia al ser divulgadas.  **3**  Grado en Ingeniería Informática  Grado en Ingeniería del Software  Grado en Ingeniería de Computadores  Luis Hernández Yáñez  Facultad de Informática  Universidad Complutense  Fundamentos de la programación  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II  Tipos, valores y variables 227  Conversión de tipos 232  Tipos declarados por el usuario 236  Tipos enumerados 238  Entrada/Salida  con archivos de texto 248  Lectura de archivos de texto 253  Escritura en archivos de texto 266  Flujo de ejecución 272  Selección simple 276  Operadores lógicos 282  Anidamiento de if 286  Condiciones 290  Selección múltiple 293  La escala if‐else‐if 295  La instrucción switch 302  Repetición 313  El bucle while 316  El bucle for 321  Bucles anidados 331  Ámbito y visibilidad 339  Secuencias 349  Recorrido de secuencias 355  Secuencias calculadas 363  Búsqueda en secuencias 370  Arrays de datos simples 374  Uso de variables arrays 379  Recorrido de arrays 382  Búsqueda en arrays 387  Arrays no completos 393  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 227  Luis Hernández Yáñez  *Tipo*  Conjunto de valores con sus posibles operaciones  *Valor*  Conjunto de bits interpretados como de un tipo concreto  *Variable (o constante)*  Cierta memoria con nombre para valores de un tipo  *Declaración*  Instrucción que identifica un nombre  *Definición*  Declaración que asigna memoria a una variable o constante  Fundamentos de la programación: Tipos e instrucciones II Página 228  Luis Hernández Yáñez  Memoria suficiente para su tipo de valores  short int i = 3;  int j = 9;  char c = 'a';  double x = 1.5;  El significado de los bits depende del tipo de la variable:  00000000 00000000 00000000 01111000  Interpretado como int es el entero 120  Interpretado como char (sólo 01111000) es el carácter 'x'  Fundamentos de la programación: Tipos e instrucciones II Página 229  i 3  j 9  c a  x 1.5  Luis Hernández Yáñez   Simples   Estándar: int, float, double, char, bool  Conjunto de valores predeterminado   Definidos por el usuario: *enumerados*  Conjunto de valores definido por el programador   Estructurados (Tema 5)   Colecciones homogéneas: *arrays*  Todos los elementos de la colección de un mismo tipo   Colecciones heterogéneas: *estructuras*  Elementos de la colección de tipos distintos  Fundamentos de programación: Tipos e instrucciones II Página 230  Luis Hernández Yáñez  Con sus posibles modificadores:  [unsigned] [short] int  long long int  float  [long] double  char  bool  Fundamentos de la programación: Tipos e instrucciones II Página 231  Definición de variables:  *tipo nombre* [ = *expresión*] [, ...];  Definición de constantes con nombre:  const *tipo nombre* = *expresión*;  long int  int  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 232  Luis Hernández Yáñez  *Promoción de tipos*  Dos operandos de tipos distintos:  El valor del tipo *menor* se promociona al tipo *mayor*  short int i = 3;  int j = 2;  double a = 1.5, b;  b = a + i \* j;  Fundamentos de la programación: Tipos e instrucciones II Página 233  **long double**  **double**  **float**  **long int**  **int**  **short int**  Promoción  Valor 3 short int (2 bytes)  int (4 bytes)  Valor 6 int (4 bytes)  double (8 bytes)  b = a + 3 \* 2;  b = 1.5 + 6;  Luis Hernández Yáñez  Conversión segura:  De un tipo menor a un tipo mayor  short int  int  long int  ...  Conversión no segura:  De un tipo mayor a un tipo menor  int entero = 1234;  char caracter;  caracter = entero; // Conversión no segura  Menor memoria: Pérdida de información en la conversión  Fundamentos de la programación: Tipos e instrucciones II Página 234  **long double**  **double**  **float**  **long int**  **int**  **short int**  Luis Hernández Yáñez  Fuerzan una conversión de tipo:  *tipo*(*expresión*)  El valor resultante de la *expresión* se trata como un valor del *tipo*  int a = 3, b = 2;  cout << a / b; // Muestra 1 (división entera)  cout << double(a) / b; // Muestra 1.5 (división real)  Tienen la mayor prioridad  Fundamentos de la programación: Tipos e instrucciones II Página 235  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 236  Luis Hernández Yáñez  Describimos los valores de las variables del tipo  typedef *descripción nombre\_de\_tipo*;  Fundamentos de la programación: Tipos e instrucciones II Página 237  Nombres de tipos propios:  t minúscula seguida de una o varias palabras capitalizadas  Los colorearemos en naranja, para remarcar que son tipos  typedef *descripción* tMiTipo;  typedef *descripción* tMoneda;  typedef *descripción* tTiposDeCalificacion;  *Identificador válido*  *Declaración de tipo frente a definición de variable*  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 238  Luis Hernández Yáñez  Enumeración del conjunto de valores posibles para las variables:  enum { *símbolo1*, *símbolo2*, ..., *símboloN* }  enum { centimo, dos\_centimos, cinco\_centimos,  diez\_centimos, veinte\_centimos,  medio\_euro, euro }  Valores literales que pueden tomar las variables (en amarillo)  Fundamentos de la programación: Tipos e instrucciones II Página 239  enum  ,  { Identificador }  Luis Hernández Yáñez  typedef *descripción nombre\_de\_tipo*;  Elegimos un nombre para el tipo: tMoneda  typedef enum { centimo, dos\_centimos, cinco\_centimos,  diez\_centimos, veinte\_centimos,  medio\_euro, euro } tMoneda;  Fundamentos de la programación: Tipos e instrucciones II Página 240  En el ámbito de la declaración, se reconoce un nuevo tipo tMoneda  tMoneda moneda1, moneda2;  Cada variable de ese tipo contendrá alguno de los símbolos  moneda1 = dos\_centimos;  moneda2 = euro;  moneda1 dos\_centimos  moneda2 euro  *descripción*  Mejoran la legibilidad  (Internamente se usan enteros)  Luis Hernández Yáñez  typedef enum { enero, febrero, marzo, abril, mayo,  junio, julio, agosto, septiembre, octubre,  noviembre, diciembre } tMes;  tMes mes;  Lectura de la variable mes:  cin >> mes;  Se espera un valor entero  No se puede escribir directamente enero o junio  Y si se escribe la variable en la pantalla:  cout << mes;  Se verá un número entero   Código de entrada/salida específico  Fundamentos de la programación: Tipos e instrucciones II Página 241  Luis Hernández Yáñez  typedef enum { enero, febrero, marzo, abril, mayo, junio, julio,  agosto, septiembre, octubre, noviembre, diciembre } tMes;  Fundamentos de la programación: Tipos e instrucciones II Página 242  int op;  cout << " 1 ‐ Enero" << endl;  cout << " 2 ‐ Febrero" << endl;  cout << " 3 ‐ Marzo" << endl;  cout << " 4 ‐ Abril" << endl;  cout << " 5 ‐ Mayo" << endl;  cout << " 6 ‐ Junio" << endl;  cout << " 7 ‐ Julio" << endl;  cout << " 8 ‐ Agosto" << endl;  cout << " 9 ‐ Septiembre" << endl;  cout << "10 ‐ Octubre" << endl;  cout << "11 ‐ Noviembre" << endl;  cout << "12 ‐ Diciembre" << endl;  cout << "Numero de mes: ";  cin >> op;  tMes mes = tMes(op ‐ 1);  Luis Hernández Yáñez  typedef enum { enero, febrero, marzo, abril, mayo, junio, julio,  agosto, septiembre, octubre, noviembre, diciembre } tMes;  Fundamentos de la programación: Tipos e instrucciones II Página 243  if (mes == enero) {  cout << "enero";  }  if (mes == febrero) {  cout << "febrero";  }  if (mes == marzo) {  cout << "marzo";  }  ...  if (mes == diciembre) {  cout << "diciembre";  }  También podemos utilizar una instrucción switch  Luis Hernández Yáñez  Conjunto de valores ordenado (posición en la enumeración)  typedef enum { lunes, martes, miercoles, jueves,  viernes, sabado, domingo } tDiaSemana;  tDiaSemana dia;  ...  if (dia == jueves)...  bool noLaborable = (dia >= sabado);  No admiten operadores de incremento y decremento  Emulación con moldes:  int i = int(dia); // ¡dia no ha de valer domingo!  i++;  dia = tDiaSemana(i);  Fundamentos de la programación: Tipos e instrucciones II Página 244  lunes < martes < miercoles < jueves  < viernes < sabado < domingo  Luis Hernández Yáñez  #include <iostream>  using namespace std;  typedef enum { enero, febrero, marzo, abril, mayo,  junio, julio, agosto, septiembre, octubre,  noviembre, diciembre } tMes;  typedef enum { lunes, martes, miercoles, jueves,  viernes, sabado, domingo } tDiaSemana;  string cadMes(tMes mes);  string cadDia(tDiaSemana dia);  int main() {  tDiaSemana hoy = lunes;  int dia = 21;  tMes mes = octubre;  int anio = 2013;  ...  Fundamentos de la programación: Tipos e instrucciones II Página 245  Si los tipos se usan en varias funciones,  los declaramos antes de los prototipos  Luis Hernández Yáñez  // Mostramos la fecha  cout << "Hoy es: " << cadDia(hoy) << " " << dia  << " de " << cadMes(mes) << " de " << anio  << endl;  cout << "Pasada la medianoche..." << endl;  dia++;  int i = int(hoy);  i++;  hoy = tDiaSemana(i);  // Mostramos la fecha  cout << "Hoy es: " << cadDia(hoy) << " " << dia  << " de " << cadMes(mes) << " de " << anio  << endl;  return 0;  }  Fundamentos de la programación: Tipos e instrucciones II Página 246  Luis Hernández Yáñez  string cadMes(tMes mes) {  string cad;  if (mes == enero) {  cad = "enero";  }  if (mes == febrero) {  cad = "febrero";  }  ...  if (mes == diciembre) {  cad = "diciembre";  }  return cad;  }  string cadDia(tDiaSemana dia);  string cad;  if (dia == lunes) {  cad = "lunes";  }  if (dia == martes) {  cad = "martes";  }  ...  if (dia == domingo) {  cad = "domingo";  }  return cad;  }  Fundamentos de la programación: Tipos e instrucciones II Página 247  fechas.cpp  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 248  Luis Hernández Yáñez  Datos del programa: en la memoria principal (volátil)  Medios (dispositivos) de almacenamiento permanente:  — Discos magnéticos fijos (internos) o portátiles (externos)  — Cintas magnéticas  — Discos ópticos (CD, DVD, BlueRay)  — Memorias USB  …  Mantienen la información en archivos  Secuencias de datos  Fundamentos de la programación: Tipos e instrucciones II Página 249  Luis Hernández Yáñez  Archivo de texto: secuencia de caracteres  Archivo binario: contiene una secuencia de códigos binarios  Los archivos se manejan en los programas por medio de *flujos*  Archivos de texto: *flujos de texto*  Similar a la E/S por consola  (Más adelante veremos el uso de archivos binarios)  Fundamentos de la programación: Tipos e instrucciones II Página 250  **A0 25 2F 04 D6 FF 00 27 6C CA 49 07 5F A4 …**  **T o t a l : 1 2 3 . 4**  **A …**  (Códigos representados en notación hexadecimal)  Luis Hernández Yáñez  Textos dispuestos en sucesivas líneas  Carácter de fin de línea entre línea y línea (Intro)  Posiblemente varios datos en cada línea  Ejemplo: Compras de los clientes  En cada línea, NIF del cliente, unidades compradas, precio  unitario y descripción de producto, separados por espacio  12345678F 2 123.95 Reproductor de DVD↲  00112233A 1 218.4 Disco portátil↲  32143567J 3 32 Memoria USB 16Gb↲  76329845H 1 134.5 Modem ADSL↲  ...  Normalmente terminan con un dato especial (*centinela*)  Por ejemplo, un NIF que sea X  Fundamentos de la programación: Tipos e instrucciones II Página 251  Luis Hernández Yáñez   Lectura del archivo: flujo de entrada   Escritura en el archivo: flujo de salida  No podemos leer y escribir en un mismo flujo  Un flujo de texto se puede utilizar para lectura o para escritura:  — Flujos (archivos) de entrada: variables de tipo ifstream  — Flujos (archivos) de salida : variables de tipo ofstream  Biblioteca fstream (sin espacio de nombres)  Fundamentos de la programación: Tipos e instrucciones II Página 252  #include <fstream>  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 253  Luis Hernández Yáñez  *Flujos de texto de entrada*  Para leer de un archivo de texto:  Declara una variable de tipo ifstream  Asocia la variable con el archivo de texto (*apertura del archivo*)  Realiza las operaciones de lectura  Desliga la variable del archivo de texto (*cierre el archivo*)  Fundamentos de la programación: Tipos e instrucciones II Página 254  **1**  **2**  **3**  **4**  ifstream  Luis Hernández Yáñez  *Apertura del archivo*  Conecta la variable con el archivo de texto del dispositivo  *flujo*.open(*cadena\_literal*);  ifstream archivo;  archivo.open("abc.txt");  if (archivo.is\_open()) ...  *Cierre del archivo*  Desconecta la variable del archivo de texto del dispositivo  *flujo*.close();  archivo.close();  Fundamentos de la programación: Tipos e instrucciones II Página 255  *¡El archivo debe existir!*  is\_open():  true si el archivo  se ha podido abrir  false en caso contrario  Luis Hernández Yáñez  *Operaciones de lectura*   Extractor (>>) archivo >> variable;  Salta primero los espacios en blanco (espacio, tab, Intro, ...)  Datos numéricos: lee hasta el primer carácter no válido  Cadenas (string): lee hasta el siguiente espacio en blanco   archivo.get(*c*)  Lee el siguiente carácter en la variable c, sea el que sea   getline(*archivo*, *cadena*)  Lee en la *cadena* todos los caracteres que queden en la línea  Incluidos los espacios en blanco  Hasta el siguiente salto de línea (descartándolo)  Con los archivos no tiene efecto la función sync()  Fundamentos de la programación: Tipos e instrucciones II Página 256  Luis Hernández Yáñez  *¿Qué debo leer?*   Un número  Usa el extractor archivo >> num;   Un carácter (sea el que sea)  Usa la función get() archivo.get(c);   Una cadena sin espacios  Usa el extractor archivo >> cad;   Una cadena posiblemente con espacios  Usa la función getline() getline(archivo, cad);  Fundamentos de la programación: Tipos e instrucciones II Página 257  Luis Hernández Yáñez  *¿Dónde queda pendiente la entrada?*   Número leído con el extractor  En el primer carácter no válido (inc. espacios en blanco)   Carácter leído con get()  En el siguiente carácter (inc. espacios en blanco)   Una cadena leída con el extractor  En el siguiente espacio en blanco   Una cadena leída con la función getline()  Al principio de la siguiente línea  Fundamentos de la programación: Tipos e instrucciones II Página 258  Luis Hernández Yáñez  Programa  string nif, producto;  int unidades;  double precio;  char aux;  Fundamentos de la programación: Tipos e instrucciones II Página 259  **7**  **6**  **5**  **4**  **3**  **2**  **1**  Flujo de entrada  archivo  **1** ifstream archivo;  **2** archivo.open("compras.txt"); // Apertura  **3** archivo >> nif >> unidades >> precio;  getline(archivo, producto);  **4** archivo.close(); // Cierre  Luis Hernández Yáñez  archivo >> nif;  archivo >> unidades;  archivo >> precio;  getline(archivo, producto);  Fundamentos de la programación: Tipos e instrucciones II Página 260  producto Reproductor de DVD precio 123.95  nif 12345678F unidades 2  12345678F 2 123.95 Reproductor de DVD  Espacio  El extractor salta los espacios  getline() no salta espacios  Luis Hernández Yáñez  archivo >> nif;  archivo >> unidades;  archivo >> precio;  archivo.get(aux); // Salta el espacio en blanco  getline(archivo, producto);  Fundamentos de la programación: Tipos e instrucciones II Página 261  producto Reproductor de DVD precio 123.95  nif 12345678F unidades 2  12345678F 2 123.95 Reproductor de DVD  Sin espacio  Leemos el espacio  (no hacemos nada con él)  Luis Hernández Yáñez  Cada línea, datos de una compra  Mostrar el total de cada compra  unidades x precio más IVA (21%)  Final: "X" como NIF  Bucle de procesamiento:   Cada paso del bucle (ciclo) procesa una línea (compra)   Podemos usar las mismas variables en cada ciclo  *Leer primer NIF*  *Mientras el NIF no sea X:*  *Leer unidades, precio y descripción*  *Calcular y mostrar el total*  *Leer el siguiente NIF*  Fundamentos de la programación: Tipos e instrucciones II Página 262  Luis Hernández Yáñez  #include <iostream>  #include <string>  using namespace std;  #include <fstream>  #include <iomanip> // Formato de salida  int main() {  const int IVA = 21;  string nif, producto;  int unidades;  double precio, neto, total, iva;  char aux;  ifstream archivo;  int contador = 0;  archivo.open("compras.txt"); // Apertura  ...  Fundamentos de la programación: Tipos e instrucciones II Página 263  leer.cpp  Luis Hernández Yáñez  if (archivo.is\_open()) { // Existe el archivo  archivo >> nif; // Primer NIF  while (nif != "X") {  archivo >> unidades >> precio;  archivo.get(aux); // Salta el espacio  getline(archivo, producto);  contador++;  neto = unidades \* precio;  iva = neto \* IVA / 100;  total = neto + iva;  cout << "Compra " << contador << ".‐" << endl;  cout << " " << producto << ": " << unidades  << " x " << fixed << setprecision(2)  << precio << " = " << neto << " ‐ I.V.A.: "  << iva << " ‐ Total: " << total << endl;  archivo >> nif; // Siguiente NIF  } ...  Fundamentos de la programación: Tipos e instrucciones II Página 264  Luis Hernández Yáñez  archivo.close(); // Cierre  }  else {  cout << "ERROR: No se ha podido abrir el archivo"  << endl;  }  return 0;  }  Fundamentos de la programación: Tipos e instrucciones II Página 265  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 266  Luis Hernández Yáñez  *Flujos de texto de salida*  Para crear un archivo de texto y escribir en él:  Declara una variable de tipo ofstream  Asocia la variable con el archivo de texto (*crea el archivo*)  Realiza las escrituras por medio del operador << (insertor)  Desliga la variable del archivo de texto (*cierra el archivo*)  Fundamentos de la programación: Tipos e instrucciones II Página 267  **1**  **2**  **3**  **4**  *¡Atención!*  Si el archivo ya existe, se borra todo lo que hubiera  *¡Atención!*  Si no se cierra el archivo se puede perder información  ofstream  Luis Hernández Yáñez  int valor = 999;  Fundamentos de la programación: Tipos e instrucciones II Página 268  **2**  **1**  **!**  **a**  **l**  **o**  **H**  **X**  Flujo de salida  archivo  Programa  **1** ofstream archivo;  **2** archivo.open("output.txt"); // Apertura  **3** archivo << 'X' << " Hola! " << 123.45  << endl << valor << "Bye!";  **4** archivo.close(); // Cierre  Luis Hernández Yáñez  #include <iostream>  #include <string>  using namespace std;  #include <fstream>  int main() {  string nif, producto;  int unidades;  double precio;  char aux;  ofstream archivo;  archivo.open("output.txt"); // Apertura (creación)  cout << "NIF del cliente (X para terminar): ";  cin >> nif;  ...  Fundamentos de la programación: Tipos e instrucciones II Página 269  escribir.cpp  Luis Hernández Yáñez  while (nif != "X") {  // Queda pendiente el Intro anterior...  cin.get(aux); // Leemos el Intro  cout << "Producto: ";  getline(cin, producto);  cout << "Unidades: ";  cin >> unidades;  cout << "Precio: ";  cin >> precio;  // Escribimos los datos en una línea del archivo...  // Con un espacio de separación entre ellos  archivo << nif << " " << unidades << " "  << precio << " " << producto << endl;  cout << "NIF del cliente (X para terminar): ";  cin >> nif;  }  ...  Fundamentos de la programación: Tipos e instrucciones II Página 270  Luis Hernández Yáñez  // Escribimos el centinela final...  archivo << "X";  archivo.close(); // Cierre del archivo  return 0;  }  Fundamentos de la programación: Tipos e instrucciones II Página 271  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 272  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 273  {  double oper1, oper2, prod;  cout << "Operando 1: ";  cin >> oper1;  cout << "Operando 2: ";  ...  cout << "Producto: " << prod;  return 0;  }  F l u j o d e e j e c u c i ó n  *Instrucción N*  *Instrucción 3*  *Instrucción 2*  *Instrucción 1*  Luis Hernández Yáñez  *Uno entre dos o más caminos de ejecución*  Selección simple (2 caminos) Selección múltiple (> 2 caminos)  Fundamentos de la programación: Tipos e instrucciones II Página 274  *Instrucción T Instrucción F*  *Condición*  true false  true  true  true  true  false  false  false  false  *Diagramas de flujo*  if  if‐else‐if  switch  Luis Hernández Yáñez  *Repetir la ejecución de una o más instrucciones*  Acumular, procesar colecciones, ...  Fundamentos de la programación: Tipos e instrucciones II Página 275  *Código*  *¿Iterar?*  Sí No  *Inicialización*  while for  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 276  Luis Hernández Yáñez  *La instrucción* if  if (*condición*) {  *códigoT*  }  [else {  *códigoF*  }]  *condición*: expresión bool  Cláusula else opcional  Fundamentos de la programación: Tipos e instrucciones II Página 277  BloqueT BloqueF  Condición  true false  *Opcional*  Luis Hernández Yáñez  int num;  cin >> num;  if (num < 0) {  cout << "Negativo";  }  else {  cout << "Positivo";  }  cout << endl;  Fundamentos de la programación: Tipos e instrucciones II Página 278  cout << "Negativo";  true  cout << "Positivo";  false  cout << endl;  num < 0  cin >> num;  signo.cpp  Luis Hernández Yáñez  int num;  cin >> num;  if (num < 0) {  cout << "Negativo";  }  else {  cout << "Positivo";  }  cout << endl;  Fundamentos de la programación: Tipos e instrucciones II Página 279  cout << endl;  cout << "Positivo";  false  num < 0  cin >> num;  \_ \_  129  Positivo  \_  **num 12?9**  Luis Hernández Yáñez  true  cout << "Negativo";  int num;  cin >> num;  if (num < 0) {  cout << "Negativo";  }  else {  cout << "Positivo";  }  cout << endl;  Fundamentos de la programación: Tipos e instrucciones II Página 280  cout << endl;  num < 0  cin >> num;  ‐5  Negativo  \_  **num ‐?5**  Luis Hernández Yáñez  División entre dos números protegida frente a intento de división por 0  #include <iostream>  using namespace std;  int main() {  double numerador, denominador, resultado;  cout << "Numerador: ";  cin >> numerador;  cout << "Denominador: ";  cin >> denominador;  if (denominador == 0) {  cout << "Imposible dividir entre 0!";  }  else {  resultado = numerador / denominador;  cout << "Resultado: " << resultado << endl;  }  return 0;  }  Fundamentos de la programación: Tipos e instrucciones II Página 281  división.cpp  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 282  Luis Hernández Yáñez  Se aplican a valores bool (*condiciones*)  El resultado es de tipo bool  Fundamentos de la programación: Tipos e instrucciones II Página 283  ! NO Monario  && Y Binario  || O Binario  **Operadores (prioridad)**  ...  !  \* / %  + ‐  < <= > >=  == !=  &&  ||  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 284  !  **true false**  **false true**  && **true false**  **true true false**  **false false false**  || **true false**  **true true true**  **false true false**  NO (*Not*) Y (*And*) O (*Or*)  bool cond1, cond2, resultado;  int a = 2, b = 3, c = 4;  resultado = !(a < 5); // !(2 < 5)  !true  false  cond1 = (a \* b + c) >= 12; // 10 >= 12  false  cond2 = (a \* (b + c)) >= 12; // 14 >= 12  true  resultado = cond1 && cond2; // false && true  false  resultado = cond1 || cond2; // false || true  true  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 285  #include <iostream>  using namespace std;  int main()  {  int num;  cout << "Introduce un número entre 1 y 10: ";  cin >> num;  if ((num >= 1) && (num <= 10)) {  cout << "Número dentro del intervalo de valores válidos";  }  else {  cout << "Número no válido!";  }  return 0;  }  condiciones.cpp  ((num >= 1) && (num <= 10))  ((num > 0) && (num < 11))  ((num >= 1) && (num < 11))  ((num > 0) && (num <= 10))  ¡Encierra las condiciones  simples entre paréntesis!  Condiciones equivalentes  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 286  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 287  diasmes.cpp  int mes, anio, dias;  cout << "Número de mes: ";  cin >> mes;  cout << "Año: ";  cin >> anio;  if (mes == 2) {  if (bisiesto(mes, anio)) {  dias = 29;  }  else {  dias = 28;  }  }  else {  if ((mes == 1) || (mes == 3) || (mes == 5) || (mes == 7)  || (mes == 8) || (mes == 10) || (mes == 12)) {  dias = 31;  }  else {  dias = 30;  }  }  Luis Hernández Yáñez  *Calendario Gregoriano*: bisiesto si divisible por 4, excepto el último  de cada siglo (divisible por 100), salvo que sea divisible por 400  Fundamentos de la programación: Tipos e instrucciones II Página 288  bool bisiesto(int mes, int anio) {  bool esBisiesto;  if ((anio % 4) == 0) { // Divisible por 4  if (((anio % 100) == 0) && ((anio % 400) != 0)) {  // Pero último de siglo y no múltiplo de 400  esBisiesto = false;  }  else {  esBisiesto = true; // Año bisiesto  }  }  else {  esBisiesto = false;  }  return esBisiesto;  }  Luis Hernández Yáñez  Cada else se asocia al if anterior más cercano sin asociar (mismo bloque)  if (*condición1*) {  if (*condición2*) {...}  else {...}  }  else {  if (*condición3*) {  if (*condición4*) {...}  if (*condición5*) {...}  else {...}  }  else { ...  Fundamentos de la programación: Tipos e instrucciones II Página 289  La sangría ayuda a asociar los else con sus if  Una mala sangría puede confundir  if (x > 0) {  if (y > 0) {...}  else {...}  if (x > 0) {  if (y > 0) {...}  else {...}  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 290  Luis Hernández Yáñez  • Condición simple: Expresión lógica (true/false)  Sin operadores lógicos  num < 0  car == 'a'  isalpha(car)  12  • Condición compuesta:  Combinación de condiciones simples y operadores lógicos  !isalpha(car)  (num < 0) || (car == 'a')  (num < 0) && ((car == 'a') || !isalpha(car))  Fundamentos de la programación: Tipos e instrucciones II Página 291  No confundas el operador de igualdad (==)  con el operador de asignación (=).  Compatibilidad con el lenguaje C:  0 es equivalente a false  Cualquier valor distinto de 0 es equivalente a true  Luis Hernández Yáñez  *Shortcut Boolean Evaluation*  true || *X*  true  (n == 0) || (x >= 1.0 / n)  Si n es 0: ¿división por cero? (segunda condición)  Como la primera sería true: ¡no se evalúa la segunda!  false && *X*  false  (n != 0) && (x < 1.0 / n)  Si n es 0: ¿división por cero? (segunda condición)  Como la primera sería false: ¡no se evalúa la segunda!  Fundamentos de la programación: Tipos e instrucciones II Página 292  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 293  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 294  true  true  true  true  false  false  false  false  if‐else‐if  switch  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 295  Luis Hernández Yáñez  Ejemplo:  Calificación (en letras)  de un estudiante en base  a su nota numérica (0‐10)  Fundamentos de la programación: Tipos e instrucciones II Página 296  true  cout << "MH"  true  cout << "SB"  true  cout << "NT"  true  cout << "AP"  false  cout << "SS"  false  >= 5  false  >= 7  false  >= 9  == 10  Si nota == 10 entonces MH  si no, si nota >= 9 entonces SB  si no, si nota >= 7 entonces NT  si no, si nota >= 5 entonces AP  si no SS  Luis Hernández Yáñez  double nota;  cin >> nota;  if (nota == 10) {  cout << "MH";  }  else {  if (nota >= 9) {  cout << "SB";  }  else {  if (nota >= 7) {  cout << "NT";  }  else {  if (nota >= 5) {  cout << "AP";  }  else {  cout << "SS";  }  }  }  }  Fundamentos de la programación: Tipos e instrucciones II Página 297  double nota;  cin >> nota;  if (nota == 10) {  cout << "MH";  }  else if (nota >= 9) {  cout << "SB";  }  else if (nota >= 7) {  cout << "NT";  }  else if (nota >= 5) {  cout << "AP";  }  else {  cout << "SS";  }    nota.cpp  Luis Hernández Yáñez  *¡Cuidado con el orden de las condiciones!*  double nota;  cin >> nota;  if (nota < 5) { cout << "SS"; }  else if (nota < 7) { cout << "AP"; }  else if (nota < 9) { cout << "NT"; }  else if (nota < 10) { cout << "SB"; }  else { cout << "MH"; }  double nota;  cin >> nota;  if (nota >= 5) { cout << "AP"; }  else if (nota >= 7) { cout << "NT"; }  else if (nota >= 9) { cout << "SB"; }  else if (nota == 10) { cout << "MH"; }  else { cout << "SS"; }  Fundamentos de la programación: Tipos e instrucciones II Página 298      Sólo muestra AP o SS  ¡No se ejecutan nunca!  Luis Hernández Yáñez  *Simplificación de las condiciones*  if (nota == 10) { cout << "MH"; }  else if ((nota < 10) && (nota >= 9)) { cout << "SB"; }  else if ((nota < 9) && (nota >= 7)) { cout << "NT"; }  else if ((nota < 7) && (nota >= 5)) { cout << "AP"; }  else if (nota < 5) { cout << "SS"; }  Fundamentos de la programación: Tipos e instrucciones II Página 299  0 5 7 9 10  MH  SS AP NT SB  Siempre true: ramas else  Si no es 10, es menor que 10  Si no es >= 9, es menor que 9  Si no es >= 7, es menor que 7  …  true && X  X  if (nota == 10) { cout << "MH"; }  else if (nota >= 9) {cout << "SB"; }  else if (nota >= 7) { cout << "NT"; }  else if (nota >= 5) { cout << "AP"; }  else { cout << "SS"; }  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 300  Si num == 4 entonces Muy alto  Si num == 3 entonces Alto  Si num == 2 entonces Medio  Si num == 1 entonces Bajo  nivel.cpp  #include <iostream>  using namespace std;  int main() {  int num;  cout << "Introduce el nivel: ";  cin >> num;  if (num == 4) {  cout << "Muy alto" << endl;  }  else if (num == 3) {  cout << "Alto" << endl;  }  else if (num == 2) {  cout << "Medio" << endl;  }  else if (num == 1) {  cout << "Bajo" << endl;  }  else {  cout << "Valor no válido" << endl;  }  return 0;  }  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 301  if (num == 4) { cout << "Muy alto" << endl; }  else if (num == 3) { cout << "Alto" << endl; }  else if (num == 2) { cout << "Medio" << endl; }  else if (num == 1) { cout << "Bajo" << endl; }  else cout << "Valor no válido" << endl; }  if (num == 4) cout << "Muy alto";  else if (num == 3) cout << "Alto";  else if (num == 2) cout << "Medio";  else if (num == 1) cout << "Bajo";  else cout << "Valor no válido";  cout << endl;  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 302  Luis Hernández Yáñez  *Selección entre valores posibles de una expresión*  Fundamentos de la programación: Tipos e instrucciones II Página 303  switch (*expresión*) {  case *constante1*:  {  *código1*  }  [break;]  case *constante2*:  {  *código2*  }  [break;]  ...  case *constanteN*:  {  *códigoN*  }  [break;]  [default:  {  *códigoDefault*  }]  }  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 304  nivel2.cpp  switch (num) {  case 4:  {  cout << "Muy alto";  }  break;  case 3:  {  cout << "Alto";  }  break;  case 2:  {  cout << "Medio";  }  break;  case 1:  {  cout << "Bajo";  }  break;  default:  {  cout << "Valor no válido";  }  }  Si num == 4  Muy alto  Si num == 3  Alto  Si num == 2  Medio  Si num == 1  Bajo  Luis Hernández Yáñez  *Interrumpe* el switch; continúa en la instrucción que le siga  Num: 3  Alto  Fundamentos de la programación: Tipos e instrucciones II Página 305  switch (num) {  ...  case 3:  {  cout << "Alto";  }  break;  case 2:  {  cout << "Medio";  }  break;  ...  }  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 306  switch (num) {  ...  case 3:  {  cout << "Alto";  }  case 2:  {  cout << "Medio";  }  case 1:  {  cout << "Bajo";  }  default:  {  cout << "Valor no válido";  }  }  Num: 3  Alto  Medio  Bajo  Valor no válido  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 307  num==4 Muy alto  true  Alto  true  Medio  true  Bajo  true  num==3  false  num==2  false  num==1  false  false  No válido  default  break;  break;  break;  break;  Sin break;  Sin break;  Sin break;  Sin break;  Luis Hernández Yáñez  int menu() {  int op = ‐1; // Cualquiera no válida  while ((op < 0) || (op > 4)) {  cout << "1 ‐ Nuevo cliente" << endl;  cout << "2 ‐ Editar cliente" << endl;  cout << "3 ‐ Baja cliente" << endl;  cout << "4 ‐ Ver cliente" << endl;  cout << "0 ‐ Salir" << endl;  cout << "Opción: ";  cin >> op;  if ((op < 0) || (op > 4)) {  cout << "¡Opción no válida!" << endl;  }  }  return op;  }  Fundamentos de la programación: Tipos e instrucciones II Página 308  cliente  4 ‐ Ver cliente  0 ‐ Salir  Opción: 3  1 ‐ Nuevo cliente  2 ‐ Editar cliente  3 ‐ Baja cliente  4 ‐ Ver cliente  0 ‐ Salir  Opción: 5  ¡Opción no válida!  1 ‐ Nuevo cliente  2 ‐ Editar cliente  3 ‐ Baja  Luis Hernández Yáñez  int opcion;  ...  opcion = menu();  switch (opcion) {  case 1:  {  cout << "En la opción 1..." << endl;  }  break;  case 2:  {  cout << "En la opción 2..." << endl;  }  break;  case 3:  {  cout << "En la opción 3..." << endl;  }  break;  case 4:  {  cout << "En la opción 4..." << endl;  } // En la última no necesitamos break  }  Fundamentos de la programación: Tipos e instrucciones II Página 309  Luis Hernández Yáñez  int opcion;  ...  opcion = menu();  while (opcion != 0) {  switch (opcion) {  case 1:  {  cout << "En la opción 1..." << endl;  }  break;  ...  case 4:  {  cout << "En la opción 4..." << endl;  }  } // switch  ...  opcion = menu();  } // while  Fundamentos de la programación: Tipos e instrucciones II Página 310  Luis Hernández Yáñez  int nota; // Sin decimales  cout << "Nota (0‐10): ";  cin >> nota;  switch (nota) {  case 0:  case 1:  case 2:  case 3:  case 4:  {  cout << "Suspenso";  }  break; // De 0 a 4: SS  case 5:  case 6:  {  cout << "Aprobado";  }  break; // 5 o 6: AP  case 7:  case 8:  {  cout << "Notable";  }  break; // 7 u 8: NT  case 9:  case 10:  {  cout << "Sobresaliente";  }  break; // 9 o 10: SB  default:  {  cout << "¡No válida!";  }  }  Fundamentos de la programación: Tipos e instrucciones II Página 311  nota2.cpp  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 312  typedef enum { enero, febrero, marzo, abril, mayo, junio,  julio, agosto, septiembre, octubre, noviembre, diciembre }  tMes;  tMes mes;  ...  switch (mes) {  case enero:  {  cout << "enero";  }  break;  case febrero:  {  cout << "febrero";  }  break;  ...  case diciembre:  {  cout << "diciembre";  }  }  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 313  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 314  *Cuerpo*  Sí No  Bucles while y for  *¿Iterar?*  *Inicialización*  Luis Hernández Yáñez   Número de iteraciones condicionado (*recorrido variable*):  — Bucle while  while (*condición*) *cuerpo*  Ejecuta el *cuerpo* mientras la *condición* sea true  — Bucle do‐while  Comprueba la condición al final (lo veremos más adelante)   Número de iteraciones prefijado (*recorrido fijo*):  — Bucle for  for (*inicialización*; *condición*; *paso*) *cuerpo*  Ejecuta el *cuerpo* mientras la *condición* sea true  Se usa una variable contadora entera  Fundamentos de la programación: Tipos e instrucciones II Página 315  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 316  Luis Hernández Yáñez  *Mientras la condición sea cierta, ejecuta el cuerpo*  while (*condición*) {  *cuerpo*  }  int i = 1; // Inicialización de la variable i  while (i <= 100) {  cout << i << endl;  i++;  }  Muestra los números del 1 al 100  Fundamentos de la programación: Tipos e instrucciones II Página 317  while.cpp  Condición al principio del bucle  Luis Hernández Yáñez  int i = 1;  while (i <= 100) {  cout << i << endl;  i++;  }  Fundamentos de la programación: Tipos e instrucciones II Página 318  *Cuerpo*  *Condición*  true false  false  cout << i << endl;  true  i++  i <= 100  i = 1  i **?123**  \_  \_1  1  2  100  3  **4**  99  **190901**  Luis Hernández Yáñez  *¿Y si la condición es falsa al comenzar?*  No se ejecuta el cuerpo del bucle ninguna vez  int op;  cout << "Introduce la opción: ";  cin >> op;  while ((op < 0) || (op > 4)) {  cout << "¡No válida! Inténtalo otra vez" << endl;  cout << "Introduce la opción: ";  cin >> op;  }  Si el usuario introduce un número entre 0 y 4:  No se ejecuta el cuerpo del bucle  Fundamentos de la programación: Tipos e instrucciones II Página 319  Luis Hernández Yáñez  *Primer entero cuyo cuadrado es mayor que 1.000*  #include <iostream>  using namespace std;  int main() {  int num = 1;  while (num \* num <= 1000) {  num++;  }  cout << "1er. entero con cuadrado mayor que 1.000: "  << num << endl;  return 0;  }  Fundamentos de la programación: Tipos e instrucciones II Página 320  *¡Ejecuta el programa para*  *saber cuál es ese número!*  primero.cpp  Recorre la *secuencia* de números 1, 2, 3, 4, 5, ...  Empezamos en 1  Incrementamos en 1  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 321  sumamedia.cpp  #include <iostream>  using namespace std;  int main() {  double num, suma = 0, media = 0;  int cont = 0;  cout << "Introduce un número (0 para terminar): ";  cin >> num;  while (num != 0) { // 0 para terminar  suma = suma + num;  cont++;  cout << "Introduce un número (0 para terminar): ";  cin >> num;  }  if (cont > 0) {  media = suma / cont;  }  cout << "Suma = " << suma << endl;  cout << "Media = " << media << endl;  return 0;  }  Recorre la *secuencia*  de números introducidos  Leemos el primero  Leemos el siguiente  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 322  Luis Hernández Yáñez  *Número de iteraciones prefijado*  Variable contadora que determina el número de iteraciones:  for ([int] *var* = *ini*; *condición*; *paso*) *cuerpo*  La *condición* compara el valor de *var* con un valor final  El *paso* incrementa o decrementa el valor de *var*  El valor de *var* debe ir aproximándose al valor final  for (int i = 1; i <= 100; i++)...  for (int i = 100; i >= 1; i‐‐)...  Tantos ciclos como valores toma la variable contadora  Fundamentos de la programación: Tipos e instrucciones II Página 323  1, 2, 3, 4, 5, ..., 100  100, 99, 98, 97, ..., 1  Luis Hernández Yáñez  for (*inicialización*; *condición*; *paso*) *cuerpo*  for (int i = 1; i <= 100; i++) {  cout << i;  }  Fundamentos de la programación: Tipos e instrucciones II Página 324  false  cout << i;  true  i++  i <= 100  i = 1  Luis Hernández Yáñez  for (int i = 1; i <= 100; i++) {  cout << i << endl;  }  Fundamentos de la programación: Tipos e instrucciones II Página 325  \_  1  **i ?**  true false  cout << i << endl;  i++  **12**  2  **3**  ...  99  **100**  100  **101**  3  i <= 100  i = 1  for1.cpp  Luis Hernández Yáñez  *La variable contadora*  El *paso* no tiene porqué ir de uno en uno:  for (int i = 1; i <= 100; i = i + 2)  cout << i << endl;  Este bucle for muestra los números impares de 1 a 99  *Garantía de terminación*  Todo bucle debe terminar su ejecución  Bucles for: la variable contadora debe converger al valor final  Fundamentos de la programación: Tipos e instrucciones II Página 326  *Muy importante*  El cuerpo del bucle NUNCA debe alterar el valor del contador  for2.cpp  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 327  suma.cpp  #include <iostream>  using namespace std;  long long int suma(int n);  int main() {  int num;  cout << "Número final: ";  cin >> num;  if (num > 0) { // El número debe ser positivo  cout << "La suma de los números entre 1 y "  << num << " es: " << suma(num);  }  return 0;  }  long long int suma(int n) {  long long int total = 0;  for (int i = 1; i <= n; i++) {  total = total + i;  }  return total;  }  Recorre la *secuencia* de números  1, 2, 3, 4, 5, ..., *n*  Luis Hernández Yáñez  *¿Incremento/decremento prefijo o postfijo?*  Es indiferente  Estos dos bucles producen el mismo resultado:  for (int i = 1; i <= 100; i++) ...  for (int i = 1; i <= 100; ++i) ...  *Bucles infinitos*  for (int i = 1; i <= 100; i‐‐) ...  1 0 ‐1 ‐2 ‐3 ‐4 ‐5 ‐6 ‐7 ‐8 ‐9 ‐10 ‐11 ...  Cada vez más lejos del valor final (100)  Es un error de diseño/programación  Fundamentos de la programación: Tipos e instrucciones II Página 328  Luis Hernández Yáñez  *Declarada en el propio bucle*  for (int i = 1; ...)  Sólo se conoce en el cuerpo del bucle (su ámbito)  No se puede usar en instrucciones que sigan al bucle  *Declarada antes del bucle*  int i;  for (i = 1; ...)  Se conoce en el cuerpo del bucle y después del mismo  Ámbito externo al bucle  Fundamentos de la programación: Tipos e instrucciones II Página 329  Luis Hernández Yáñez  Los bucles for se pueden reescribir como bucles condicionados  for (int i = 1; i <= 100; i++) *cuerpo*  Es equivalente a:  La inversa no es siempre posible:  int i;  cin >> i;  while (i != 0) {  *cuerpo*  cin >> i;  }  Fundamentos de la programación: Tipos e instrucciones II Página 330  int i = 1;  while (i <= 100) {  *cuerpo*  i++;  }  *¿Bucle* for *equivalente?*  *¡No sabemos cuántos números*  *introducirá el usuario!*  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 331  Luis Hernández Yáñez  Un bucle for en el cuerpo de otro bucle for  Cada uno con su propia variable contadora:  for (int i = 1; i <= 100; i++) {  for (int j = 1; j <= 5; j++) {  *cuerpo*  }  }  Para cada valor de i  el valor de j varía entre 1 y 5  j *varía más rápido* que i  Fundamentos de la programación: Tipos e instrucciones II Página 332  i j  1 1  1 2  1 3  1 4  1 5  2 1  2 2  2 3  2 4  2 5  3 1  ...  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 333  1 x 1 = 1  1 x 2 = 2  1 x 3 = 3  1 x 4 = 4  ...  1 x 10 = 10  2 x 1 = 2  2 x 2 = 4  ...  10 x 7 = 70  10 x 8 = 80  10 x 9 = 90  10 x 10 = 100  tablas.cpp  #include <iostream>  using namespace std;  #include <iomanip>  int main() {  for (int i = 1; i <= 10; i++) {  for (int j = 1; j <= 10; j++) {  cout << setw(2) << i << " x "  << setw(2) << j << " = "  << setw(3) << i \* j << endl;  }  }  return 0;  }  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 334  tablas2.cpp  #include <iostream>  using namespace std;  #include <iomanip>  int main() {  for (int i = 1; i <= 10; i++) {  cout << "Tabla del " << i << endl;  cout << "‐‐‐‐‐‐‐‐‐‐‐‐‐‐" << endl;  for (int j = 1; j <= 10; j++) {  cout << setw(2) << i << " x "  << setw(2) << j << " = "  << setw(3) << i \* j << endl;  }  cout << endl;  }  return 0;  }  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 335  #include <iostream>  using namespace std;  #include <iomanip>  int menu(); // 1: Tablas de multiplicación; 2: Sumatorio  long long int suma(int n); // Sumatorio  int main() {  int opcion = menu();  while (opcion != 0) {  switch (opcion) {  case 1:  {  for (int i = 1; i <= 10; i++) {  for (int j = 1; j <= 10; j++) {  cout << setw(2) << i << " x "  << setw(2) << j << " = "  << setw(3) << i \* j << endl;  }  }  }  break; ...  menú.cpp  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 336  case 2:  {  int num = 0;  while (num <= 0) {  cout << "Hasta (positivo)? ";  cin >> num;  }  cout << "La suma de los números del 1 al "  << num << " es: " << suma(num) << endl;  }  } // switch  opcion = menu();  } // while (opcion != 0)  return 0;  }  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 337  int menu() {  int op = ‐1;  while ((op < 0) || (op > 2)) {  cout << "1 ‐ Tablas de multiplicar" << endl;  cout << "2 ‐ Sumatorio" << endl;  cout << "0 ‐ Salir" << endl;  cout << "Opción: " << endl;  cin >> op;  if ((op < 0) || (op > 2)) {  cout << "¡Opción no válida!" << endl;  }  }  return op;  }  long long int suma(int n) {  long long int total = 0;  for (int i = 1; i <= n; i++) {  total = total + i;  }  return total;  }  Luis Hernández Yáñez  while (opcion != 0) {  ...  for (int i = 1; i <= 10; i++) {  for (int j = 1; j <= 10; j++) {  ...  }  }  while (num <= 0) {  ...  }  for (int i = 1; i <= n; i++) {  ...  }  while ((op < 0) || (op > 2)) {  ...  }  }  Fundamentos de la programación: Tipos e instrucciones II Página 338  menu()  suma()  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 339  Luis Hernández Yáñez  Cada bloque crea un nuevo ámbito:  int main() {  double d = ‐1, suma = 0;  int cont = 0;  while (d != 0) {  cin >> d;  if (d != 0) {  suma = suma + d;  cont++;  }  }  cout << "Suma = " << suma << endl;  cout << "Media = " << suma / cont << endl;  return 0;  }  Fundamentos de la programación: Tipos e instrucciones II Página 340  3 ámbitos anidados  Luis Hernández Yáñez  Un identificador se conoce  en el ámbito en el que está declarado  (a partir de su instrucción de declaración)  y en los subámbitos posteriores  Fundamentos de la programación: Tipos e instrucciones II Página 341  Luis Hernández Yáñez  Ámbito de la variable d  int main() {  double d;  if (...) {  int cont = 0;  for (int i = 0; i <= 10; i++) {  ...  }  }  char c;  if (...) {  double x;  ...  }  return 0;  }  Fundamentos de la programación: Tipos e instrucciones II Página 342  Luis Hernández Yáñez  int main() {  double d;  if (...) {  int cont = 0;  for (int i = 0; i <= 10; i++) {  ...  }  }  char c;  if (...) {  double x;  ...  }  return 0;  }  Ámbito de la variable cont  Fundamentos de la programación: Tipos e instrucciones II Página 343  Luis Hernández Yáñez  int main() {  double d;  if (...) {  int cont = 0;  for (int i = 0; i <= 10; i++) {  ...  }  }  char c;  if (...) {  double x;  ...  }  return 0;  }  Ámbito de la variable i  Fundamentos de la programación: Tipos e instrucciones II Página 344  Luis Hernández Yáñez  Ámbito de la variable c  int main() {  double d;  if (...) {  int cont = 0;  for (int i = 0; i <= 10; i++) {  ...  }  }  char c;  if (...) {  double x;  ...  }  return 0;  }  Fundamentos de la programación: Tipos e instrucciones II Página 345  Luis Hernández Yáñez  Ámbito de la variable x  int main() {  double d;  if (...) {  int cont = 0;  for (int i = 0; i <= 10; i++) {  ...  }  }  char c;  if (...) {  double x;  ...  }  return 0;  }  Fundamentos de la programación: Tipos e instrucciones II Página 346  Luis Hernández Yáñez  Si en un subámbito se declara  un identificador con idéntico nombre  que uno ya declarado en el ámbito,  el del subámbito *oculta* al del ámbito  (no es visible)  Fundamentos de la programación: Tipos e instrucciones II Página 347  Luis Hernández Yáñez  int main() {  int i, x;  if (...) {  int i = 0;  for(int i = 0; i <= 10; i++) {  ...  }  }  char c;  if (...) {  double x;  ...  }  return 0;  }  Fundamentos de la programación: Tipos e instrucciones II Página 348  Oculta , en su ámbito, a la i anterior  Oculta , en su ámbito, a la i anterior  Oculta , en su ámbito, a la x anterior  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 349  Luis Hernández Yáñez  Sucesión de elementos de un mismo tipo que se acceden linealmente  Fundamentos de la programación: Tipos e instrucciones II Página 350  *elemento secuencia*  *secuencia*  (Secuencia vacía)  1 34 12 26 4 87 184 52  Comienza en un *primer* elemento (si no está vacía)  A cada elemento le sigue otra secuencia (vacía, si es el *último*)  *Acceso secuencial (lineal)*  Se comienza siempre accediendo al primer elemento  Desde un elemento sólo se puede acceder a su elemento siguiente  (*sucesor*), si es que existe  Todos los elementos, de un mismo tipo  *elemento*  o  Luis Hernández Yáñez  No tratamos secuencias infinitas: siempre hay un último elemento   Secuencias explícitas:  — Sucesión de datos de un dispositivo (teclado, disco, sensor, ...)   Secuencias calculadas:  — Fórmula de recurrencia que determina el elemento siguiente   Listas (*más adelante*)  Secuencias explícitas que manejaremos:  Datos introducidos por el teclado o leídos de un archivo  Con un elemento especial al final de la secuencia (*centinela*)  1 34 12 26 4 87 184 52 ‐1  Fundamentos de la programación: Tipos e instrucciones II Página 351  Luis Hernández Yáñez   Secuencia explícita leída de archivo:  — Detectar la marca de final de archivo (Eof ‐ *End of file*)  — Detectar un valor centinela al final   Secuencia explícita leída del teclado:  — Preguntar al usuario si quiere introducir un nuevo dato  — Preguntar al usuario primero cuántos datos va a introducir  — Detectar un valor centinela al final  Valor *centinela*:  Valor especial al final que no puede darse en la secuencia  (Secuencia de números positivos  centinela: cualquier negativo)  Fundamentos de la programación: Tipos e instrucciones II Página 352  12 4 37 23 8 19 83 63 2 35 17 76 15 ‐1  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 353  12 4 37 23 8 19 83 63 ‐1 35 17 76 15  Último elemento  No se procesan  Debe haber algún valor que no sea un elemento válido  Secuencias numéricas:  Si se permite cualquier número, no hay centinela posible  Cadenas de caracteres:  ¿Caracteres especiales (no imprimibles)?  En realidad el valor centinela es parte de la secuencia,  pero su significado es especial y no se procesa como el resto  Significa que se ha alcanzado el final de la secuencia  (*Incluso aunque haya elementos posteriores*)  Luis Hernández Yáñez  Tratamiento de los elementos uno a uno desde el primero  *Recorrido*  Un mismo tratamiento para todos los elementos de la secuencia  Ej.‐ Mostrar los elementos de una secuencia, sumar los números  de una secuencia, ¿par o impar cada número de una secuencia?, ...  Termina al llegar al final de la secuencia  *Búsqueda*  Recorrido de la secuencia hasta encontrar un elemento buscado  Ej.‐ Localizar el primer número que sea mayor que 1.000  Termina al localizar el primer elemento que cumple la condición  o al llegar al final de la secuencia (*no encontrado*)  Fundamentos de la programación: Tipos e instrucciones II Página 354  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 355  Luis Hernández Yáñez  *Un mismo tratamiento a todos los elementos*  *Inicialización*  *Mientras no se llegue al final de la secuencia:*  *Obtener el siguiente elemento*  *Procesar el elemento*  *Finalización*  Al empezar se obtiene el primer elemento de la secuencia  En los siguientes pasos del bucle se van obteniendo  los siguientes elementos de la secuencia  Fundamentos de la programación: Tipos e instrucciones II Página 356  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 357  false  Procesar elemento  Obtener elemento  true  Finalización  ¿Al final?  Inicialización  No sabemos cuántos  elementos hay   No podemos  implementar con for  Luis Hernández Yáñez  *Implementación con* while  *Inicialización*  *Obtener el primer elemento*  *Mientras no sea el centinela:*  *Procesar el elemento*  *Obtener el siguiente elemento*  *Finalización*  Fundamentos de la programación: Tipos e instrucciones II Página 358  false  Procesar elemento  Obtener siguiente  true  Finalización  ¿Centinela?  Inicialización  Obtener 1º  Luis Hernández Yáñez  *Secuencia de números positivos*  Siempre se realiza al menos una lectura  Centinela: ‐1  double d, suma = 0;  cout << "Valor (‐1 termina): ";  cin >> d;  while (d != ‐1) {  suma = suma + d;  cout << "Valor (‐1 termina): ";  cin >> d;  }  cout << "Suma = " << suma << endl;  Fundamentos de la programación: Tipos e instrucciones II Página 359  Inicialización  Primer elemento  Mientras no el centinela  Procesar elemento  Siguiente elemento  Finalización  Luis Hernández Yáñez  *Longitud de una secuencia de caracteres*  Centinela: carácter punto (.)  Fundamentos de la programación: Tipos e instrucciones II Página 360  longitud.cpp  int longitud() {  int l = 0;  char c;  cout << "Texto terminado en punto: ";  cin >> c; // Obtener primer carácter  while (c != '.') { // Mientras no el centinela  l++; // Procesar  cin >> c; // Obtener siguiente carácter  }  return l;  }  Luis Hernández Yáñez  *¿Cuántas veces aparece un carácter en una cadena?*  Centinela: asterisco (\*)  Fundamentos de la programación: Tipos e instrucciones II Página 361  cont.cpp  char buscado, c;  int cont = 0;  cout << "Carácter a buscar: ";  cin >> buscado;  cout << "Cadena: ";  cin >> c;  while (c != '\*') {  if (c == buscado) {  cont++;  }  cin >> c;  }  cout << buscado << " aparece " << cont  << " veces.";  Mientras no el centinela  Procesar elemento  Siguiente elemento  Primer elemento  Luis Hernández Yáñez  *Suma de los números de la secuencia*  Centinela: 0  Fundamentos de la programación: Tipos e instrucciones II Página 362  suma2.cpp  int sumaSecuencia() {  double d, suma = 0;  ifstream archivo; // Archivo de entrada (lectura)  archivo.open("datos.txt");  if (archivo.is\_open()) {  archivo >> d; // Obtener el primero  while (d != 0) { // Mientras no sea el centinela  suma = suma + d; // Procesar el dato  archivo >> d; // Obtener el siguiente  }  archivo.close();  }  return suma;  }  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 363  Luis Hernández Yáñez  *Recurrencia: ei+1 = ei + 1 e1 = 1*  1 2 3 4 5 6 7 8 ...  Suma de los números de la secuencia calculada:  int main() {  int num;  cout << "N = ";  cin >> num;  cout << "Sumatorio:" << suma(num);  return 0;  }  long long int suma(int n) {  int sumatorio = 0;  for (int i = 1; i <= n; i++) {  sumatorio = sumatorio + i;  }  return sumatorio;  }  Fundamentos de la programación: Tipos e instrucciones II Página 364  sumatorio.cpp  Último elemento de la secuencia: n  Luis Hernández Yáñez  long long int suma(int n) {  int sumatorio = 0;  for (int i = 1; i <= n; i++) {  sumatorio = sumatorio + i;  }  ...  Fundamentos de la programación: Tipos e instrucciones II Página 365  n **?**  sumatorio **0**  i **1**  i <= n  **01**  **2**  **5**  **3**  **3**  **6**  **4**  **10**  **5**  **15**  **6 12345 1234 123 12 1**  Secuencia  false  true  i = i + 1;  sumatorio += i;  int i = 1;  sumatorio = 0;  Luis Hernández Yáñez  *Definición*  Fi = Fi‐1 + Fi‐2  F1 = 0  F2 = 1  0 1 1 2 3 5 8 13 21 34 55 89 ...  *¿Fin de la secuencia?*  Primer número de Fibonacci mayor que un número dado  Ese número de Fibonacci actúa como centinela  Si num es 50, la secuencia será:  0 1 1 2 3 5 8 13 21 34  Fundamentos de la programación: Tipos e instrucciones II Página 366  Luis Hernández Yáñez  *Recorrido de la secuencia calculada*  Fundamentos de la programación: Tipos e instrucciones II Página 367  fibonacci.cpp  *¿Demasiados comentarios?*  Para no oscurecer el código, mejor una explicación al principio  int num, fib, fibMenos2 = 0, fibMenos1 = 1; // 1º y 2º  fib = fibMenos2 + fibMenos1; // Calculamos el tercero  cout << "Hasta: ";  cin >> num;  if (num >= 1) { // Ha de ser entero positivo  cout << "0 1 "; // Los dos primeros son <= num  while (fib <= num) { // Mientras no mayor que num  cout << fib << " ";  fibMenos2 = fibMenos1; // Actualizamos anteriores  fibMenos1 = fib; // para obtener...  fib = fibMenos2 + fibMenos1; // ... el siguiente  }  Fundamentos de la programación: Tipos e instrucciones II Página 367  fibonacci.cpp  *¿Demasiados comentarios?*  Para no oscurecer el código, mejor una explicación al principio  int num, fib, fibMenos2 = 0, fibMenos1 = 1; // 1º y 2º  fib = fibMenos2 + fibMenos1; // Calculamos el tercero  cout << "Hasta: ";  cin >> num;  if (num >= 1) { // Ha de ser entero positivo  cout << "0 1 "; // Los dos primeros son <= num  while (fib <= num) { // Mientras no mayor que num  cout << fib << " ";  fibMenos2 = fibMenos1; // Actualizamos anteriores  fibMenos1 = fib; // para obtener...  fib = fibMenos2 + fibMenos1; // ... el siguiente  }  }  Luis Hernández Yáñez  El bucle calcula adecuadamente la secuencia:  while (fib <= num) {  cout << fib << " ";  fibMenos2 = fibMenos1;  fibMenos1 = fib;  fib = fibMenos2 + fibMenos1;  }  Fundamentos de la programación: Tipos e instrucciones II Página 368  num 1?00  fib ?1  fibMenos1 ?1  fibMenos2 ?0  0 1 1  1  2  2  2  3  3  2  3  5  5 ...  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 369  Luis Hernández Yáñez  *Localización del primer elemento con una propiedad*  *Inicialización*  *Mientras no se encuentre el elemento*  *y no se esté al final de la secuencia:*  *Obtener el siguiente elemento*  *Comprobar si el elemento satisface la condición*  *Finalización*  *(tratar el elemento encontrado o indicar que no se ha encontrado)*  Fundamentos de la programación: Tipos e instrucciones II Página 370  Elemento que se busca: satisfará una condición  Dos condiciones de terminación del bucle: se encuentra / al final  Variable lógica que indique si se ha encontrado  Luis Hernández Yáñez  *Localización del primer elemento con una propiedad*  false  ¿Encontrado?  Obtener elemento  true  Finalización  Fundamentos de la programación: Tipos e instrucciones II Página 371  Inicialización / encontrado = false;  ¿Al final o  encontrado?  Luis Hernández Yáñez  *Implementación con* while  *Inicialización*  *Obtener el primer elemento*  *Mientras ni encontrado ni el centinela:*  *Obtener el siguiente elemento*  *Finalización (¿encontrado?)*  Fundamentos de la programación: Tipos e instrucciones II Página 372  false  Obtener siguiente  true  Finalización  ¿Encontrado  o centinela?  Obtener 1º  Inicialización  Luis Hernández Yáñez  *Primer número mayor que uno dado*  Centinela: ‐1  double d, num;  bool encontrado = false;  cout << "Encontrar primero mayor que: ";  cin >> num;  cout << "Siguiente (‐1 para terminar): ";  cin >> d; // Obtener el primer elemento  while ((d != ‐1) && !encontrado) {  // Mientras no sea el centinela y no se encuentre  if (d > num) { // ¿Encontrado?  encontrado = true;  }  else {  cout << "Siguiente (‐1 para terminar): ";  cin >> d; // Obtener el siguiente elemento  }  }  Fundamentos de la programación: Tipos e instrucciones II Página 373  busca.cpp  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 374  Luis Hernández Yáñez  *Colecciones homogéneas*  Un mismo tipo de dato para varios elementos:   Notas de los estudiantes de una clase   Ventas de cada día de la semana   Temperaturas de cada día del mes  ...  En lugar de declarar *N* variables...  Fundamentos de la programación: Tipos e instrucciones II Página 375  125.40  vLun  76.95  vMar  328.80  vMie  254.62  vJue  435.00  vVie  164.29  vSab  0.00  vDom  ventas 125.40 76.95 328.80 254.62 435.00 164.29 0.00  0 1 2 3 4 5 6  ... declaramos una tabla de *N* valores:  Índices  Luis Hernández Yáñez  *Estructura secuencial*  Cada elemento se encuentra en una posición (*índice*):   Los índices son enteros positivos   El índice del primer elemento siempre es 0   Los índices se incrementan de uno en uno  *Acceso directo*  A cada elemento se accede a través de su índice:  ventas[4] accede al 5º elemento (contiene el valor 435.00)  cout << ventas[4];  ventas[4] = 442.75;  Fundamentos de la programación: Tipos e instrucciones II Página 376  ventas 125.40 76.95 328.80 254.62 435.00 164.29 0.00  0 1 2 3 4 5 6  Datos de un mismo tipo base:  Se usan como cualquier variable  Luis Hernández Yáñez  *Declaración de tipos de arrays*  typedef *tipo\_base nombre\_tipo*[*tamaño*];  Ejemplos:  typedef double tTemp[7];  typedef short int tDiasMes[12];  typedef char tVocales[5];  typedef double tVentas[31];  typedef tMoneda tCalderilla[15]; // Enumerado tMoneda  Fundamentos de la programación: Tipos e instrucciones II Página 377  *Recuerda:* Adoptamos el convenio de comenzar  los nombres de tipo con una t minúscula, seguida  de una o varias palabras, cada una con su inicial en mayúscula  Luis Hernández Yáñez  *Declaración de variables arrays*  *tipo nombre*;  Ejemplos:  Fundamentos de la programación: Tipos e instrucciones II Página 378  NO se inicializan los elementos automáticamente  tempMax ? ? ? ? ? ? ?  0 1 2 3 4 5 6  typedef double tTemp[7];  typedef short int tDiasMes[12];  typedef char tVocales[5];  typedef double tVentas[31];  ventasFeb ? ? ? ? ? ? ? ? ? ? ? ? ? ... ?  0 1 2 3 4 5 6 7 8 9 10 11 12 30  vocales ? ? ? ? ?  0 1 2 3 4  diasMes ? ? ? ? ? ? ? ? ? ? ? ?  0 1 2 3 4 5 6 7 8 9 10 11  tTemp tempMax;  tDiasMes diasMes;  tVocales vocales;  tVentas ventasFeb;  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 379  Luis Hernández Yáñez  *nombre*[*índice*]  Cada elemento se accede a través de su índice (posición en el array)  tVocales vocales;  5 elementos, índices de 0 a 4:  vocales[0] vocales[1] vocales[2] vocales[3] vocales[4]  Procesamiento de cada elemento:  Como cualquier otra variable del tipo base  cout << vocales[4];  vocales[3] = 'o';  if (vocales[i] == 'e') ...  Fundamentos de la programación: Tipos e instrucciones II Página 380  vocales 'a' 'e' 'i' 'o' 'u'  0 1 2 3 4  typedef char tVocales[5];  Luis Hernández Yáñez  ¡IMPORTANTE!  ¡No se comprueba si el índice es correcto!  *¡Es responsabilidad del programador!*  const int Dim = 100;  typedef double tVentas[Dim];  tVentas ventas;  Índices válidos: enteros entre 0 y Dim‐1  ventas[0] ventas[1] ventas[2] ... ventas[98] ventas[99]  ¿Qué es ventas[100]? ¿O ventas[‐1]? ¿O ventas[132]?  ¡Memoria de alguna otra variable del programa!  Fundamentos de la programación: Tipos e instrucciones II Página 381  Define los tamaños de los arrays con constantes  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 382  Luis Hernández Yáñez  Arrays: tamaño fijo  Bucle de recorrido fijo (for)  Ejemplo: Media de un array de temperaturas  const int Dias = 7;  typedef double tTemp[Dias];  tTemp temp;  double media, total = 0;  ...  for (int i = 0; i < Dias; i++) {  total = total + temp[i];  }  media = total / Dias;  Fundamentos de la programación: Tipos e instrucciones II Página 383  Luis Hernández Yáñez  tTemp temp;  double media, total = 0;  ...  for (int i = 0; i < Dias; i++) {  total = total + temp[i];  }  Memoria  Dias **7**  temp[0] **12.40**  temp[1] **10.96**  temp[2] **8.43**  temp[3] **11.65**  temp[4] **13.70**  temp[5] **13.41**  temp[6] **14.07**  media **?**  total **0.00**  i **0**  **12.40**  **1**  **23.36**  **2**  **31.79**  **3**  **43.44**  **4**  **84.62**  **7**  Fundamentos de la programación: Tipos e instrucciones II Página 384  12.40 10.96 8.43 11.65 13.70 13.41 14.07  0 1 2 3 4 5 6  true false  total+=temp[i]  i++  i<Dias  i = 0  ...  Luis Hernández Yáñez  Los usuarios usan de 1 a 7 para numerar los días  La interfaz debe aproximarse a los usuarios,  aunque internamente se usen los índices de 0 a 6  #include <iostream>  using namespace std;  const int Dias = 7;  typedef double tTemp[Dias];  double media(const tTemp temp);  int main() {  tTemp temp;  for (int i = 0; i < Dias; i++) { // Recorrido del array  cout << "Temperatura del día " << i + 1 << ": ";  cin >> temp[i];  }  cout << "Temperatura media: " << media(temp) << endl;  return 0;  }  ...  Fundamentos de la programación: Tipos e instrucciones II Página 385  mediatemp.cpp  Luis Hernández Yáñez  double media(const tTemp temp) {  double med, total = 0;  for (int i = 0; i < Dias; i++) { // Recorrido del array  total = total + temp[i];  }  med = total / Dias;  return med;  }  Fundamentos de la programación: Tipos e instrucciones II Página 386  Los arrays se pasan a las funciones como constantes  Las funciones no pueden devolver arrays  Luis Hernández Yáñez  const int Cuantas = 15;  typedef enum { centimo, dos\_centimos, cinco\_centimos,  diez\_centimos, veinte\_centimos, medio\_euro, euro } tMoneda;  typedef tMoneda tCalderilla[Cuantas];  string aCadena(tMoneda moneda);  // Devuelve la cadena correspondiente al valor de moneda  tCalderilla bolsillo; // Exactamente llevo Cuantas monedas  bolsillo[0] = euro;  bolsillo[1] = cinco\_centimos;  bolsillo[2] = medio\_euro;  bolsillo[3] = euro;  bolsillo[4] = centimo;  ...  for (int moneda = 0; moneda < Cuantas; moneda++)  cout << aCadena(bolsillo[moneda]) << endl;  Fundamentos de la programación: Tipos e instrucciones II Página 387  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 388  Luis Hernández Yáñez  *¿Qué día las ventas superaron los 1.000 €?*  const int Dias = 365; // Año no bisiesto  typedef double tVentas[Dias];  int busca(const tVentas ventas) {  // Índice del primer elemento mayor que 1000 (‐1 si no hay)  bool encontrado = false;  int ind = 0;  while ((ind < Dias) && !encontrado) { // Esquema de búsqueda  if (ventas[ind] > 1000) {  encontrado = true;  }  else {  ind++;  }  }  if (!encontrado) {  ind = ‐1;  }  return ind;  }  Fundamentos de la programación: Tipos e instrucciones II Página 389  buscaarray.cpp  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 390  Luis Hernández Yáñez  La capacidad de un array no puede ser alterada en la ejecución  El tamaño de un array es una decisión de diseño:   En ocasiones será fácil (días de la semana)   Cuando pueda variar ha de estimarse un tamaño  Ni corto ni con mucho desperdicio (posiciones sin usar)  STL (*Standard Template Library*) de C++:  Colecciones más eficientes cuyo tamaño puede variar  Fundamentos de la programación: Tipos e instrucciones II Página 391  Luis Hernández Yáñez  No se pueden copiar dos arrays (del mismo tipo) con asignación:  array2 = array1; // *¡¡¡ NO COPIA LOS ELEMENTOS !!!*  Han de copiarse los elementos uno a uno:  for (int i = 0; i < N; i++) {  array2[i] = array1[i];  }  Fundamentos de la programación: Tipos e instrucciones II Página 392  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II Página 393  Luis Hernández Yáñez  Puede que no necesitemos todas las posiciones de un array...  La dimensión del array será el máximo de elementos  Pero podremos tener menos elementos del máximo  Necesitamos un contador de elementos...  const int Max = 100;  typedef double tArray[Max];  tArray lista;  int contador = 0;  contador: indica cuántas posiciones del array se utilizan  Sólo accederemos a las posiciones entre 0 y contador‐1  Las demás posiciones no contienen información del programa  Fundamentos de la programación: Tipos e instrucciones II Página 394  Luis Hernández Yáñez  #include <iostream>  using namespace std;  #include <fstream>  const int Max = 100;  typedef double tArray[Max];  double media(const tArray lista, int cont);  int main() {  tArray lista;  int contador = 0;  double valor, med;  ifstream archivo;  archivo.open("lista.txt");  if (archivo.is\_open()) {  archivo >> valor;  while ((valor != ‐1) && (contador < Max)) {  lista[contador] = valor;  contador++;  archivo >> valor;  } ...  Fundamentos de la programación: Tipos e instrucciones II Página 395  lista.cpp  Luis Hernández Yáñez  archivo.close();  med = media(lista, contador);  cout << "Media de los elementos de la lista: " << med << endl;  }  else {  cout << "¡No se pudo abrir el archivo!" << endl;  }  return 0;  }  double media(const tArray lista, int cont) {  double med, total = 0;  for (int ind = 0; ind < cont; ind++) {  total = total + lista[ind];  }  med = total / cont;  return med;  }  Fundamentos de la programación: Tipos e instrucciones II Página 396  Sólo recorremos hasta cont‐1  Luis Hernández Yáñez  *Licencia CC (Creative Commons)*  Este tipo de licencias ofrecen algunos derechos a terceras personas  bajo ciertas condiciones.  Este documento tiene establecidas las siguientes:  Pulsa en la imagen de arriba a la derecha para saber más.  Fundamentos de la programación: Tipos e instrucciones II Página 397  Reconocimiento (*Attribution*):  En cualquier explotación de la obra autorizada por la licencia  hará falta reconocer la autoría.  No comercial (*Non commercial*):  La explotación de la obra queda limitada a usos no comerciales.  Compartir igual (*Share alike*):  La explotación autorizada incluye la creación de obras derivadas  siempre que mantengan la misma licencia al ser divulgadas.  **3A**  Grado en Ingeniería Informática  Grado en Ingeniería del Software  Grado en Ingeniería de Computadores  Luis Hernández Yáñez  Facultad de Informática  Universidad Complutense  Fundamentos de la programación  Luis Hernández Yáñez  *Expresión condicional*  Dos alternativas  — *Condición*: Expresión lógica  — *Exp1* y *Exp2*: Expresiones  Si *Condición* se evalúa a true,  el resultado es *Exp1*;  si *Condición* se evalúa a false,  el resultado es *Exp2*.  int a = 5, b = 3, c;  c = (a + b == 10) ? 2 : 3;  c = ( 8 == 10) ? 2 : 3;  c = false ? 2 : 3;  c = 3;  Fundamentos de la programación: Tipos e instrucciones II (Anexo I) Página 399  *Condición Exp1*  Operadores (prioridad)  ++ ‐‐ (postfijos)  Llamadas a funciones  Moldes  ++ ‐‐ (prefijos) !  ‐ (cambio de signo)  \* / %  + ‐  < <= > >=  == !=  &&  ||  ?:  = += ‐= \*= /= %=  ? : *Exp2*  Luis Hernández Yáñez  *Equivalencia con un* if‐else  c = (a + b == 10) ? 2 : 3;  Es equivalente a:  if (a + b == 10) c = 2;  else c = 3;  Se pueden concatenar:  cout << (nota == 10 ? "MH" : (nota >= 9 ? "SB" :  (nota >= 7 ? "NT" : (nota >= 5 ? "AP" : "SS"))))  Esto es equivalente a la escala if‐else‐if de la siguiente sección.  Fundamentos de la programación: Tipos e instrucciones II (Anexo I) Página 400  Luis Hernández Yáñez  Escala if ... else if ... equivalente  cout << (nota == 10 ? "MH" : (nota >= 9 ? "SB" :  (nota >= 7 ? "NT" : (nota >= 5 ? "AP" : "SS"))))  Si nota == 10 entonces MH  si no, si nota >= 9 entonces SB  si no, si nota >= 7 entonces NT  si no, si nota >= 5 entonces AP  si no SS  double nota;  cin >> nota;  if (nota == 10) { cout << "MH"; }  else if (nota >= 9) { cout << "SB"; }  else if (nota >= 7) { cout << "NT"; }  else if (nota >= 5) { cout << "AP"; }  else { cout << "SS"; }  Fundamentos de la programación: Tipos e instrucciones II (Anexo I) Página 401  "MH"  >= 9 "SB"  >= 7 "NT"  >= 5 "AP"  == 10  true  true  true  true  false  false  false  false  "SS"  **3E**  Grado en Ingeniería Informática  Grado en Ingeniería del Software  Grado en Ingeniería de Computadores  Luis Hernández Yáñez  Facultad de Informática  Universidad Complutense  Fundamentos de la programación  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II (Anexo II)  Recorridos 404  Un aparcamiento 405  ¿Paréntesis bien emparejados? 409  ¿Dos secuencias iguales? 412  Números primos menores que N 413  Búsquedas 417  Búsqueda de un número en un archivo 419  Búsquedas en secuencias ordenadas 420  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II (Anexo II) Página 404  Luis Hernández Yáñez  Secuencia de caracteres E y S en archivo  E = Entra un coche; S = Sale un coche  ¿Cuántos coches quedan al final de la jornada?  Varios casos, cada uno en una línea y terminado en punto  Final: línea sólo con punto  Fundamentos de la programación: Tipos e instrucciones II (Anexo II) Página 405  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II (Anexo II) Página 406  #include <iostream>  using namespace std;  #include <fstream>  int main() {  int coches;  char c;  bool terminar = false;  ifstream archivo;  archivo.open("parking.txt");  if (!archivo.is\_open()) {  cout << "¡No se ha podido abrir el archivo!" << endl;  }  else {  // Recorrido...  archivo.close();  }  return 0;  }  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II (Anexo II) Página 407  while (!terminar) {  archivo >> c;  if (c == '.') { // . como primer carácter? (centinela)  terminar = true;  }  else {  coches = 0;  while (c != '.') { // Recorrido de la secuencia  cout << c;  if (c == 'E') {  coches++;  }  else if (c == 'S') {  coches‐‐;  }  archivo >> c;  }  ...  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II (Anexo II) Página 408  if (coches >= 0) {  cout << endl << "Quedan " << coches << " coches.";  }  else {  cout << endl << "Error: Más salidas que entradas!";  }  cout << endl;  }  }  parking.cpp  Luis Hernández Yáñez  *Cada paréntesis, con su pareja*  Secuencia de caracteres terminada en # y con parejas de paréntesis:  a b ( c ( d e ) f g h ( ( i ( j k ) ) l m n ) o p ) ( r s ) #  Contador del nivel de anidamiento:  Al encontrar '(' incrementamos – Al encontrar ')' decrementamos  Al terminar, el contador deberá tener el valor 0  Errores:  — Contador ‐1: paréntesis de cierre sin uno de apertura pendiente  abc)de(fgh(ij))#  — Contador termina con un valor positivo  Más paréntesis de apertura que de cierre  Algún paréntesis sin cerrar: (a(b(cd(e)f)gh(i))jk#  Fundamentos de la programación: Tipos e instrucciones II (Anexo II) Página 409  Luis Hernández Yáñez  Un error puede interrumpir el recorrido:  char c;  int anidamiento = 0, pos = 0;  bool error = false;  cin >> c;  while ((c != '#') && !error) {  pos++;  if (c == '(') {  anidamiento++;  }  else if (c == ')') {  anidamiento‐‐;  }  if (anidamiento < 0) {  error = true;  }  if (!error) {  cin >> c;  }  }  Fundamentos de la programación: Tipos e instrucciones II (Anexo II) Página 410  Luis Hernández Yáñez  if (error) {  cout << "Error: cierre sin apertura (pos. " << pos  << ")";  }  else if (anidamiento > 0) {  cout << "Error: Apertura sin cierre";  }  else {  cout << "Correcto";  }  cout << endl;  Fundamentos de la programación: Tipos e instrucciones II (Anexo II) Página 411  parentesis.cpp  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II (Anexo II) Página 412  iguales.cpp  bool iguales() {  bool sonIguales = true;  double d1, d2;  ifstream sec1, sec2;  bool final = false;  sec1.open("secuencia1.txt");  sec2.open("secuencia2.txt");  sec1 >> d1;  sec2 >> d2; // Al menos estarán los centinelas (0)  while (sonIguales && !final) {  sonIguales = (d1 == d2);  final = ((d1 == 0) || (d2 == 0));  if (!final) {  sec1 >> d1;  sec2 >> d2;  }  }  sec1.close();  sec2.close();  return sonIguales;  }  Cambia secuencia2.txt por secuencia3.txt  y por secuencia4.txt para comprobar otros casos  Luis Hernández Yáñez  Secuencia calculada: números divisibles sólo por 1 y ellos mismos (< *N*)  Fundamentos de la programación: Tipos e instrucciones II (Anexo II) Página 413  primos.cpp  #include <iostream>  using namespace std;  bool primo(int n);  int main() {  int num, candidato;  cout << "Entero en el que parar (>1): ";  cin >> num;  if (num > 1) {  candidato = 2; // El 1 no se considera un número primo  while (candidato < num) {  cout << candidato << " "; // Mostrar número primo  candidato++;  while (!primo(candidato)) { // Siguiente primo  candidato++;  }  }  }  return 0;  }  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II (Anexo II) Página 414  bool primo(int n) {  bool esPrimo = true;  for (int i = 2; i <= n ‐ 1; i++) {  if (n % i == 0) {  esPrimo = false; // Es divisible por i  }  }  return esPrimo;  }  Luis Hernández Yáñez  Mejoras: probar sólo impares; sólo pueden ser divisibles por impares;  no pueden ser divisibles por ninguno mayor que su mitad  Fundamentos de la programación: Tipos e instrucciones II (Anexo II) Página 415  primos2.cpp  candidato = 2;  cout << candidato << " "; // Mostrar el número primo 2  candidato++; // Seguimos con el 3, que es primo  while (candidato < num) {  cout << candidato << " "; // Mostrar número primo  candidato = candidato + 2; // Sólo probamos impares  while (!primo(candidato)) { // Siguiente número primo  candidato = candidato + 2;  }  } ...  bool primo(int n) {  bool esPrimo = true;  for (int i = 3; i <= n / 2; i = i + 2) {  if (n % i == 0) {  esPrimo = false; // Es divisible por i  }  }...  Luis Hernández Yáñez  Otra mejora más: Paramos al encontrar el primer divisor  Fundamentos de la programación: Tipos e instrucciones II (Anexo II) Página 416  primos3.cpp  bool primo(int n) {  bool esPrimo = true;  int i = 3;  while ((i <= n / 2) && esPrimo) {  if (n % i == 0) {  esPrimo = false;  }i  = i + 2;  }  return esPrimo;  }  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II (Anexo II) Página 417  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II (Anexo II) Página 418  #include <iostream> buscaarch.cpp  using namespace std;  #include <fstream>  int busca(int n);  // Devuelve la línea en la que se encuentra o ‐1 si no está  int main() {  int num, linea;  cout << "Valor a localizar: ";  cin >> num;  linea = busca(num);  if (linea != ‐1) {  cout << "Encontrado (línea " << linea << ")" << endl;  }  else {  cout << "No encontrado" << endl;  }  return 0;  }  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II (Anexo II) Página 419  int busca(int n) {  int i, linea = 0;  bool encontrado = false;  ifstream archivo;  archivo.open("enteros.txt");  if (!archivo.is\_open()) {  linea = ‐1;  }  else {  archivo >> i;  while ((i != 0) && !encontrado) {  linea++;  if (i == n) {  encontrado = true;  }  archivo >> i;  }  if (!encontrado) {  linea = ‐1;  }  archivo.close();  }  return linea;  }  Centinela  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones II (Anexo II) Página 420  Luis Hernández Yáñez  Secuencia ordenada de menor a mayor:  paramos al encontrar uno mayor o igual al buscado  Los que resten serán seguro mayores: *¡no puede estar el buscado!*  Fundamentos de la programación: Tipos e instrucciones II (Anexo II) Página 421  buscaord.cpp  cout << "Valor a localizar: ";  cin >> num;  archivo >> i;  while ((i != 0) && (i < num)) {  cont++;  archivo >> i;  }  if (i == num) {  cout << "Encontrado (pos.: " << cont << ")";  }  else {  cout << "No encontrado";  }  cout << endl;  archivo.close();  Luis Hernández Yáñez  Si el elemento está: procesamiento similar a secuencias desordenadas  Fundamentos de la programación: Tipos e instrucciones II (Anexo II) Página 422  (i != 0) false  && (i < num)  true  cont++;  archivo >> i;  2 5 9 15 16 24 41 73 78 82 123 153 159 ...  **num ?**  **i ?2**  **9**  **59**  archivo >> i;  Luis Hernández Yáñez  Si el elemento no está: evitamos buscar en el resto de la secuencia  Fundamentos de la programación: Tipos e instrucciones II (Anexo II) Página 423  (i != 0) false  && (i < num)  true  cont++;  archivo >> i;  2 5 9 15 16 24 41 73 78 82 123 153 159 ...  **num ?**  **i ?2**  **10**  **59 15**  *No se procesa*  *el resto*  *de la secuencia*  archivo >> i;  Luis Hernández Yáñez  *Licencia CC (Creative Commons)*  Este tipo de licencias ofrecen algunos derechos a terceras personas  bajo ciertas condiciones.  Este documento tiene establecidas las siguientes:  Pulsa en la imagen de arriba a la derecha para saber más.  Fundamentos de la programación: Tipos e instrucciones II (Anexo II) Página 424  Reconocimiento (*Attribution*):  En cualquier explotación de la obra autorizada por la licencia  hará falta reconocer la autoría.  No comercial (*Non commercial*):  La explotación de la obra queda limitada a usos no comerciales.  Compartir igual (*Share alike*):  La explotación autorizada incluye la creación de obras derivadas  siempre que mantengan la misma licencia al ser divulgadas.  **4**  Grado en Ingeniería Informática  Grado en Ingeniería del Software  Grado en Ingeniería de Computadores  Luis Hernández Yáñez  Facultad de Informática  Universidad Complutense  Fundamentos de la programación  Luis Hernández Yáñez  Fundamentos de la programación: La abstracción procedimental  Diseño descendente: Tareas y subtareas 427  Subprogramas 434  Subprogramas y datos 441  Parámetros 446  Argumentos 451  Resultado de la función 467  Prototipos 473  Ejemplos completos 475  Funciones de operador 477  Diseño descendente (un ejemplo) 480  Precondiciones y postcondiciones 490  Luis Hernández Yáñez  Fundamentos de la programación: La abstracción procedimental Página 427  Luis Hernández Yáñez  *Refinamientos sucesivos*  Tareas que ha de realizar un programa:  Se pueden dividir en subtareas más sencillas  Subtareas:  También se pueden dividir en otras más sencillas...   Refinamientos sucesivos  Diseño en sucesivos pasos en los se amplía el detalle  Ejemplos:   Dibujar   Mostrar la cadena HOLA MAMA en letras gigantes  Fundamentos de la programación: La abstracción procedimental Página 428  Luis Hernández Yáñez  1. Dibujar  2. Dibujar  3. Dibujar  Fundamentos de la programación: La abstracción procedimental Página 429  1. Dibujar  2. Dibujar  2.1. Dibujar  2.2. Dibujar  3. Dibujar  REFINAMIENTO  Misma tarea  Luis Hernández Yáñez  1. Dibujar  2. Dibujar  2.1. Dibujar  2.2. Dibujar  3. Dibujar  Fundamentos de la programación: La abstracción procedimental Página 430  4 tareas, pero dos de ellas son iguales  Nos basta con saber cómo dibujar:  Luis Hernández Yáñez  Fundamentos de la programación: La abstracción procedimental Página 431  *Dibujar Dibujar*  *Dibujar Dibujar Dibujar*  *Dibujar*  void dibujarCirculo()  { ... }  void dibujarSecantes()  { ... }  void dibujarLinea()  { ... }  void dibujarTriangulo()  {  dibujarSecantes();  dibujarLinea();  }  int main() {  dibujarCirculo();  dibujarTriangulo();  dibujarSecantes();  return 0;  }  Luis Hernández Yáñez  Mostrar la cadena HOLA MAMA en letras gigantes  Fundamentos de la programación: La abstracción procedimental Página 432  *H O L A M A*  *Mostrar HOLA Espacio en blanco Mostrar MAMA*  *Mostrar HOLA MAMA*  *H O L A Espacio en blanco M*  Tareas básicas  Luis Hernández Yáñez  Fundamentos de la programación: La abstracción procedimental Página 433  void mostrarH() {  cout << "\* \*" << endl;  cout << "\* \*" << endl;  cout << "\*\*\*\*\*" << endl;  cout << "\* \*" << endl;  cout << "\* \*" << endl << endl;  }  void mostrarO() {  cout << "\*\*\*\*\*" << endl;  cout << "\* \*" << endl;  cout << "\* \*" << endl;  cout << "\* \*" << endl;  cout << "\*\*\*\*\*" << endl << endl;  }  void mostrarL()  { ... }  void mostrarA()  { ...}  void espaciosEnBlanco() {  cout << endl << endl << endl;  }  void mostrarM()  { ...}  int main() {  mostrarH();  mostrarO();  mostrarL();  mostrarA();  espaciosEnBlanco();  mostrarM();  mostrarA();  mostrarM();  mostrarA();  return 0;  }  Luis Hernández Yáñez  Fundamentos de la programación: La abstracción procedimental Página 434  Luis Hernández Yáñez  *Subprogramas*  *Pequeños programas dentro de otros programas*   Unidades de ejecución independientes   Encapsulan código y datos   Se comunican con otros subprogramas (datos)  *Subrutinas, procedimientos, funciones, acciones, ...*   Realizan tareas individuales del programa   Funcionalidad concreta, identificable y coherente (diseño)   Se ejecutan de principio a fin cuando se llaman (*invocan*)   Terminan devolviendo el control al punto de llamada  Fundamentos de la programación: La abstracción procedimental Página 435  Aumentan el nivel de abstracción del programa  Facilitan la prueba, la depuración y el mantenimiento  Luis Hernández Yáñez  *Flujo de ejecución*  Fundamentos de la programación: La abstracción procedimental Página 436  int main()  {  mostrarH();  mostrarO();  ...  }  void mostrarH()  {  ...  }  void mostrarO()  {  ...  }  ...          Luis Hernández Yáñez  *Subprogramas en C++*  Forma general de un subprograma en C++:  *tipo nombre*(*parámetros*) // Cabecera  {  // *Cuerpo*  }   *Tipo* de dato que devuelve el subprograma como resultado   *Parámetros* para la comunicación con el exterior   *Cuerpo*: ¡Un bloque de código!  Fundamentos de la programación: La abstracción procedimental Página 437  Luis Hernández Yáñez  *Tipos de subprogramas*  Procedimientos (*acciones*):  NO devuelven ningún resultado de su ejecución con return  Tipo: void  Llamada: instrucción independiente  mostrarH();  Funciones:  SÍ devuelven un resultado con la instrucción return  Tipo distinto de void  Llamada: dentro de cualquier expresión  x = 12 \* y + cuadrado(20) ‐ 3;  Se sustituye en la expresión por el valor que devuelve  *¡Ya venimos utilizando funciones desde el Tema 2!*  Fundamentos de la programación: La abstracción procedimental Página 438  Luis Hernández Yáñez  *Funciones*  Subprogramas de tipo distinto de void  ...  int menu()  {  int op;  cout << "1 – Editar" << endl;  cout << "2 – Combinar" << endl;  cout << "3 – Publicar" << endl;  cout << "0 – Cancelar" << endl;  cout << "Elija: ";  cin >> op;  return op;  }  Fundamentos de la programación: La abstracción procedimental Página 439  int main()  {  ...  int opcion;  opcion = menu() ;  ...  Luis Hernández Yáñez  *Procedimientos*  Subprogramas de tipo void  ...  void menu()  {  int op;  cout << "1 – Editar" << endl;  cout << "2 – Combinar" << endl;  cout << "0 – Cancelar" << endl;  cout << "Opción: ";  cin >> op;  if (op == 1) {  editar();  }  else if (op == 2) {  combinar();  }  }  Fundamentos de la programación: La abstracción procedimental Página 440  int main()  {  ...  menu();  ...  Luis Hernández Yáñez  Fundamentos de la programación: La abstracción procedimental Página 441  Luis Hernández Yáñez  *De uso exclusivo del subprograma*  *tipo nombre*(*parámetros*) // Cabecera  {  *Declaraciones locales* // Cuerpo  }   Declaraciones locales de tipos, constantes y variables  Dentro del cuerpo del subprograma   Parámetros declarados en la cabecera del subprograma  Comunicación del subprograma con otros subprogramas  Fundamentos de la programación: La abstracción procedimental Página 442  Luis Hernández Yáñez  *Datos en los programas*   Datos globales: declarados fuera de todos los subprogramas  Existen durante toda la ejecución del programa   Datos locales: declarados en algún subprograma  Existen sólo durante la ejecución del subprograma  *Ámbito y visibilidad de los datos Tema 3*  — Ámbito de los datos globales: resto del programa  Se conocen dentro de los subprogramas que siguen  — Ámbito de los datos locales: resto del subprograma  No se conocen fuera del subprograma  — Visibilidad de los datos  Datos locales a un bloque ocultan otros externos homónimos  Fundamentos de la programación: La abstracción procedimental Página 443  Luis Hernández Yáñez  #include <iostream>  using namespace std;  const int MAX = 100;  double ingresos;  ...  void proc() {  int op;  double ingresos;  ...  }  int main() {  int op;  ...  return 0;  }  Fundamentos de la programación: La abstracción procedimental Página 444  Datos globales  Datos locales a proc()  Datos locales a main()  Se conocen MAX (global), op (local)  e ingresos (local que oculta la global)  Se conocen MAX (global), op (local)  e ingresos (global)  op de proc()  es distinta  de op de main()  Luis Hernández Yáñez  *Sobre el uso de datos globales en los subprogramas*  NO SE DEBEN USAR datos globales en subprogramas   *¿Necesidad de datos externos?*  Define parámetros en el subprograma  Los datos externos se pasan como argumentos en la llamada   Uso de datos globales en los subprogramas:  Riesgo de *efectos laterales*  Modificación inadvertida de esos datos afectando otros sitios  Excepciones:   Constantes globales (valores inalterables)   Tipos globales (necesarios en varios subprogramas)  Fundamentos de la programación: La abstracción procedimental Página 445  Luis Hernández Yáñez  Fundamentos de la programación: La abstracción procedimental Página 446  Luis Hernández Yáñez  Datos de entrada, datos de salida y datos de entrada/salida  Datos de entrada: Aceptados  Subprograma que dado un número  muestra en la pantalla su cuadrado:  Datos de salida: Devueltos  Subprograma que dado un número  devuelve su cuadrado:  Datos de entrada/salida:  Aceptados y modificados  Subprograma que dada una variable  numérica la eleva al cuadrado:  Fundamentos de la programación: La abstracción procedimental Página 447  *Subprograma*  *Subprograma*  *Subprograma*  cuadrado()  x  5  cuadrado()  x y(=x2)  5  cuadrado()  x x  Luis Hernández Yáñez  *Declaración de parámetros*  Sólo dos clases de parámetros en C++:  — Sólo de entrada (*por valor*)  — De salida (sólo salida o E/S) (*por referencia* / *por variable*)  *Lista de parámetros formales*  Entre los paréntesis de la cabecera del subprograma  *tipo nombre*(*parámetros*)  Fundamentos de la programación: La abstracción procedimental Página 448  *tipo identificador*  ,  *parámetros*  De salida &  Luis Hernández Yáñez  Reciben copias de los argumentos usados en la llamada  int cuadrado(int num)  double potencia(double base, int exp)  void muestra(string nombre, int edad, string nif)  void proc(char c, int x, double a, bool b)  Reciben sus valores en la llamada del subprograma  Argumentos: Expresiones en general  Variables, constantes, literales, llamadas a función, operaciones  Se destruyen al terminar la ejecución del subprograma  *¡Atención!* Los arrays se pasan por valor como constantes:  double media(const tArray lista)  Fundamentos de la programación: La abstracción procedimental Página 449  Luis Hernández Yáñez  Misma identidad que la variable pasada como argumento  void incrementa(int &x)  void intercambia(double &x, double &y)  void proc(char &c, int &x, double &a, bool &b)  Reciben las variables en la llamada del subprograma: *¡Variables!*  Los argumentos pueden quedar modificados  *¡No usaremos parámetros por valor en las funciones!*  Sólo en procedimientos  *¡Atención!* Los arrays se pasan por referencia sin utilizar &  void insertar(tArray lista, int &contador, double item)  El argumento de lista (variable tArray) quedará modificado  Fundamentos de la programación: La abstracción procedimental Página 450  &  Puede haber tanto por valor como por referencia  Luis Hernández Yáñez  Fundamentos de la programación: La abstracción procedimental Página 451  Luis Hernández Yáñez  *nombre*(*argumentos*)  — Tantos argumentos como parámetros y en el mismo orden  — Concordancia de tipos argumento‐parámetro  — Por valor: Expresiones válidas (se pasa el resultado)  — Por referencia: *¡Sólo variables!*  Se copian los valores de las expresiones pasadas por valor  en los correspondientes parámetros  Se hacen corresponder los argumentos pasados por referencia  (variables) con sus correspondientes parámetros  Fundamentos de la programación: La abstracción procedimental Página 452  Luis Hernández Yáñez  Expresiones válidas con concordancia de tipo:  void proc(int x, double a)  proc(23 \* 4 / 7, 13.5);   double d = 3;  proc(12, d);   double d = 3;  int i = 124;  proc(i, 33 \* d);   double d = 3;  int i = 124;  proc(cuad(20) \* 34 + i, i \* d);  Fundamentos de la programación: La abstracción procedimental Página 453  Luis Hernández Yáñez  void proc(int x, double a)  { ... }  int main()  {  int i = 124;  double d = 3;  proc(i, 33 \* d);  ...  return 0;  }  Fundamentos de la programación: La abstracción procedimental Página 454  Memoria  i 124  d 3.0  **...**  **...**  x 124  a 99.0  **...**  Luis Hernández Yáñez  void proc(int &x, double &a)  { ... }  int main()  {  int i = 124;  double d = 3;  proc(i, d);  ...  return 0;  }  Fundamentos de la programación: La abstracción procedimental Página 455  Memoria  i 124  d 3.0  **...**  x  a  Luis Hernández Yáñez  Dadas las siguientes declaraciones:  int i;  double d;  void proc(int x, double &a);  *¿Qué pasos de argumentos son correctos? ¿Por qué no?*  proc(3, i, d);  proc(i, d);  proc(3 \* i + 12, d);  proc(i, 23);  proc(d, i);  proc(3.5, d);  proc(i);  Fundamentos de la programación: La abstracción procedimental Página 456   Nº de argumentos ≠ Nº de parámetros         Parámetro por referencia  ¡variable!   ¡Argumento double para parámetro int!   ¡Argumento double para parámetro int!   Nº de argumentos ≠ Nº de parámetros  Luis Hernández Yáñez  ...  void divide(int op1, int op2, int &div, int &rem) {  // Divide op1 entre op2 y devuelve el cociente y el resto  div = op1 / op2;  rem = op1 % op2;  }  int main() {  int cociente, resto;  for (int j = 1; j <= 4; j++) {  for (int i = 1; i <= 4; i++) {  divide(i, j, cociente, resto);  cout << i << " entre " << j << " da un cociente de "  << cociente << " y un resto de " << resto << endl;  }  }  return 0;  }  Fundamentos de la programación: La abstracción procedimental Página 457  Luis Hernández Yáñez  ...  void divide(int op1, int op2, int &div, int &rem) {  // Divide op1 entre op2 y devuelve el cociente y el resto  div = op1 / op2;  rem = op1 % op2;  }  int main() {  int cociente, resto;  for (int j = 1; j <= 4; j++) {  for (int i = 1; i <= 4; i++) {  divide(i, j, cociente, resto);  ...  }  }  return 0;  }  Fundamentos de la programación: La abstracción procedimental Página 458  Memoria  cociente ?  resto ?  i 1  j 1  **...**  Luis Hernández Yáñez  ...  void divide(int op1, int op2, int &div, int &rem) {  // Divide op1 entre op2 y devuelve el cociente y el resto  div = op1 / op2;  rem = op1 % op2;  }  int main() {  int cociente, resto;  for (int j = 1; j <= 4; j++) {  for (int i = 1; i <= 4; i++) {  divide(i, j, cociente, resto);  ...  }  }  return 0;  }  Fundamentos de la programación: La abstracción procedimental Página 459  Memoria  cociente ?  resto ?  i 1  j 1  **...**  div  rem  op1 **1**  op2 1  **...**  Luis Hernández Yáñez  ...  void divide(int op1, int op2, int &div, int &rem) {  // Divide op1 entre op2 y devuelve el cociente y el resto  div = op1 / op2;  rem = op1 % op2;  }  int main() {  int cociente, resto;  for (int j = 1; j <= 4; j++) {  for (int i = 1; i <= 4; i++) {  divide(i, j, cociente, resto);  ...  }  }  return 0;  }  Fundamentos de la programación: La abstracción procedimental Página 460  Memoria  cociente 1  resto 0  i 1  j 1  **...**  div  rem  op1 **1**  op2 1  **...**  Luis Hernández Yáñez  Fundamentos de la programación: La abstracción procedimental Página 461  Memoria  cociente 1  resto 0  i 1  j 1  **...**  ...  void divide(int op1, int op2, int &div, int &rem) {  // Divide op1 entre op2 y devuelve el cociente y el resto  div = op1 / op2;  rem = op1 % op2;  }  int main() {  int cociente, resto;  for (int j = 1; j <= 4; j++) {  for (int i = 1; i <= 4; i++) {  divide(i, j, cociente, resto);  ...  }  }  return 0;  }  Luis Hernández Yáñez  ...  void intercambia(double &valor1, double &valor2) {  // Intercambia los valores  double tmp; // Variable local (temporal)  tmp = valor1;  valor1 = valor2;  valor2 = tmp;  }  int main() {  double num1, num2;  cout << "Valor 1: ";  cin >> num1;  cout << "Valor 2: ";  cin >> num2;  intercambia(num1, num2);  cout << "Ahora el valor 1 es " << num1  << " y el valor 2 es " << num2 << endl;  return 0;  }  Fundamentos de la programación: La abstracción procedimental Página 462  Memoria temporal  del procedimiento  tmp ?  **...**  Memoria de main()  num1 13.6  num2 317.14  **...**  valor1  valor2  Luis Hernández Yáñez  ...  // Prototipo  void cambio(double precio, double pago, int &euros, int &cent50,  int &cent20, int &cent10, int &cent5, int &cent2, int &cent1);  int main() {  double precio, pago;  int euros, cent50, cent20, cent10, cent5, cent2, cent1;  cout << "Precio: ";  cin >> precio;  cout << "Pago: ";  cin >> pago;  cambio(precio, pago, euros, cent50, cent20, cent10, cent5, cent2,  cent1);  cout << "Cambio: " << euros << " euros, " << cent50 << " x 50c., "  << cent20 << " x 20c., " << cent10 << " x 10c., "  << cent5 << " x 5c., " << cent2 << " x 2c. y "  << cent1 << " x 1c." << endl;  return 0;  }  Fundamentos de la programación: La abstracción procedimental Página 463  Luis Hernández Yáñez  void cambio(double precio, double pago, int &euros, int &cent50,  int &cent20, int &cent10, int &cent5, int &cent2, int &cent1) {  if (pago < precio) { // Cantidad insuficiente  cout << "Error: El pago es inferior al precio" << endl;  }  else {  int cantidad = int(100.0 \* (pago ‐ precio) + 0.5);  euros = cantidad / 100;  cantidad = cambio % 100;  cent50 = cantidad / 50;  cantidad = cantidad % 50;  cent20 = cantidad / 20;  cantidad = cantidad % 20;  cent10 = cantidad / 10;  cantidad = cantidad % 10;  cent5 = cantidad / 5;  cantidad = cantidad % 5;  cent2 = cantidad / 2;  cent1 = cantidad % 2;  }  }  Fundamentos de la programación: La abstracción procedimental Página 464  Explicación en el libro de  Adams/Leestma/Nyhoff  Luis Hernández Yáñez  En los subprogramas se pueden detectar errores  Errores que impiden realizar los cálculos:  void cambio(double precio, double pago, int &euros, int &cent50,  int &cent20, int &cent10, int &cent5, int &cent2, int &cent1) {  if (pago < precio) { // Cantidad insuficiente  cout << "Error: El pago es inferior al precio" << endl;  }  ...  ¿Debe el subprograma notificar al usuario o al programa?   Mejor notificarlo al punto de llamada y allí decidir qué hacer  void cambio(double precio, double pago, int &euros, int &cent50,  int &cent20, int &cent10, int &cent5, int &cent2, int &cent1,  bool &error) {  if (pago < precio) { // Cantidad insuficiente  error = true;  }  else {  error = false;  ...  Fundamentos de la programación: La abstracción procedimental Página 465  Luis Hernández Yáñez  Al volver de la llamada se decide qué hacer si ha habido error...   ¿Informar al usuario?   ¿Volver a pedir los datos?   Etcétera  int main() {  double precio, pago;  int euros, cent50, cent20, cent10, cent5, cent2, cent1;  bool error;  cout << "Precio: ";  cin >> precio;  cout << "Pago: ";  cin >> pago;  cambio(precio, pago, euros, cent50, cent20, cent10, cent5, cent2,  cent1, error);  if (error) {  cout << "Error: El pago es inferior al precio" << endl;  }  else {  ...  Fundamentos de la programación: La abstracción procedimental Página 466  cambio.cpp  Luis Hernández Yáñez  Fundamentos de la programación: La abstracción procedimental Página 467  Luis Hernández Yáñez  *Una función ha de devolver un resultado*  La función ha de terminar su ejecución devolviendo el resultado  La instrucción return:  — Devuelve el dato que se indica a continuación como resultado  — Termina la ejecución de la función  El dato devuelto sustituye a la llamada de la función en la expresión  int cuad(int x) { int main() {  return x \* x; cout << 2 \* cuad(16);  x = x \* x;  } return 0;  }  Fundamentos de la programación: La abstracción procedimental Página 468  Esta instrucción  256  no se ejecutará nunca  Luis Hernández Yáñez  Factorial (N) = 1 x 2 x 3 x ... x (N‐2) x (N‐1) x N  long long int factorial(int n); // Prototipo  int main() {  int num;  cout << "Num: ";  cin >> num;  cout << "Factorial de " << num << ": " << factorial(num) << endl;  return 0;  }  long long int factorial(int n) {  long long int fact = 1;  if (n < 0) {  fact = 0;  }  else {  for (int i = 1; i <= n; i++) {  fact = fact \* i;  }  }  return fact;  }  Fundamentos de la programación: La abstracción procedimental Página 469  factorial.cpp  Luis Hernández Yáñez  Fundamentos de la programación: La abstracción procedimental Página 470  int compara(int val1, int val2) {  // ‐1 si val1 < val2, 0 si iguales, +1 si val1 > val2  if (val1 == val2) {  return 0;  }  else if (val1 < val2) {  return ‐1;  }  else {  return 1;  }  }  ¡3 puntos de salida!   Luis Hernández Yáñez  Fundamentos de la programación: La abstracción procedimental Página 471  int compara(int val1, int val2) {  // ‐1 si val1 < val2, 0 si iguales, +1 si val1 > val2  int resultado;  if (val1 == val2) {  resultado = 0;  }  else if (val1 < val2) {  resultado = ‐1;  }  else {  resultado = 1;  }  return resultado;  }  Punto de salida único   Luis Hernández Yáñez  Procedimientos (tipo void):  — Al encontrar la llave de cierre que termina el subprograma o  — Al encontrar una instrucción return (sin resultado)  Funciones (tipo distinto de void):  — SÓLO al encontrar una instrucción return (con resultado)  Nuestros subprogramas siempre terminarán al final:   No usaremos return en los procedimientos   Funciones: sólo un return y estará al final  Fundamentos de la programación: La abstracción procedimental Página 472  Para facilitar la depuración y el mantenimiento,  codifica los subprogramas con un único punto de salida  Luis Hernández Yáñez  Fundamentos de la programación: La abstracción procedimental Página 473  Luis Hernández Yáñez  ¿Dónde los ponemos? ¿Antes de main()? ¿Después de main()?   Los pondremos después de main()  ¿Son correctas las llamadas a subprogramas?  En main() o en otros subprogramas  — ¿Existe el subprograma?  — ¿Concuerdan los argumentos con los parámetros?  Deben estar los prototipos de los subprogramas antes de main()  Prototipo: cabecera del subprograma terminada en ;  void dibujarCirculo();  void mostrarM();  void proc(double &a);  int cuad(int x);  ...  Fundamentos de la programación: La abstracción procedimental Página 474  main() es el único subprograma  que no hay que prototipar  Luis Hernández Yáñez  #include <iostream>  using namespace std;  void intercambia(double &valor1, double &valor2); // Prototipo  int main() {  double num1, num2;  cout << "Valor 1: ";  cin >> num1;  cout << "Valor 2: ";  cin >> num2;  intercambia(num1, num2);  cout << "Ahora el valor 1 es " << num1  << " y el valor 2 es " << num2 << endl;  return 0;  }  void intercambia(double &valor1, double &valor2) {  double tmp; // Variable local (temporal)  tmp = valor1;  valor1 = valor2;  valor2 = tmp;  }  Fundamentos de la programación: La abstracción procedimental Página 475  intercambia.cpp  Asegúrate de que los prototipos  coincidan con las implementaciones  Luis Hernández Yáñez  #include <iostream>  using namespace std;  // Prototipos  long long int factorial(int n);  int sumatorio(int n);  int main() {  int num;  cout << "Num: ";  cin >> num;  cout << "Factorial de "  << num << ": "  << factorial(num)  << endl  << "Sumatorio de 1 a "  << num << ": "  << sumatorio(num)  << endl;  return 0;  }  long long int factorial(int n) {  long long int fact = 1;  if (n < 0) {  fact = 0;  }  else {  for (int i = 1; i <= n; i++) {  fact = fact \* i;  }  }  return fact;  }  int sumatorio(int n) {  int sum = 0;  for (int i = 1; i <= n; i++) {  sum = sum + i;  }  return sum;  }  Fundamentos de la programación: La abstracción procedimental Página 476  mates.cpp  Luis Hernández Yáñez  Fundamentos de la programación: La abstracción procedimental Página 477  Luis Hernández Yáñez  *Notación infija (de operador)*  *operandoIzquierdo operador operandoDerecho*  a + b  Se ejecuta el operador con los operandos como argumentos  Los operadores se implementan como funciones:  *tipo* operator*símbolo*(*parámetros*)  Si es un operador monario sólo habrá un parámetro  Si es binario habrá dos parámetros  El *símbolo* es un símbolo de operador (uno o dos caracteres):  +, ‐, \*, /, ‐‐, <<, %, ...  Fundamentos de la programación: La abstracción procedimental Página 478  Luis Hernández Yáñez  tMatriz suma(tMatriz a, tMatriz b);  tMatriz a, b, c;  c = suma(a, b);  tMatriz operator+(tMatriz a, tMatriz b);  tMatriz a, b, c;  c = a + b;  ¡La implementación será exactamente la misma!  Mayor aproximación al lenguaje matemático  Fundamentos de la programación: La abstracción procedimental Página 479  Luis Hernández Yáñez  Fundamentos de la programación: La abstracción procedimental Página 480  Luis Hernández Yáñez  Especificación inicial (Paso 0).‐  *Desarrollar un programa que haga operaciones de conversión*  *de medidas hasta que el usuario decida que no quiere hacer más*  Análisis y diseño aumentando el nivel de detalle en cada paso  *¿Qué operaciones de conversión?*  Paso 1.‐  *Desarrollar un programa que haga operaciones de conversión*  *de medidas hasta que el usuario decida que no quiere hacer más*   *Pulgadas a centímetros*   *Libras a gramos*   *Grados Fahrenheit a centígrados*   *Galones a litros*  Fundamentos de la programación: La abstracción procedimental Página 481  Luis Hernández Yáñez  Paso 2.‐  *Desarrollar un programa que muestre al usuario un menú con*  *cuatro operaciones de conversión de medidas:*   *Pulgadas a centímetros*   *Libras a gramos*   *Grados Fahrenheit a centígrados*   *Galones a litros*  *Y lea la elección del usuario y proceda con la conversión, hasta que*  *el usuario decida que no quiere hacer más*  6 grandes tareas:  Menú, cuatro funciones de conversión y main()  Fundamentos de la programación: La abstracción procedimental Página 482  Luis Hernández Yáñez  Paso 2.‐  Fundamentos de la programación: La abstracción procedimental Página 483  *Menú Pulgadas a cm. Libras a gr. ºF a ºC Galones a l.*  *Conversiones*  *main()*  Luis Hernández Yáñez  Paso 3.‐   *Menú:*  Mostrar las cuatro opciones más una para salir  Validar la entrada y devolver la elegida   *Pulgadas a centímetros:*  Devolver el equivalente en centímetros del valor en pulgadas   *Libras a gramos:*  Devolver el equivalente en gramos del valor en libras   *Grados Fahrenheit a centígrados:*  Devolver el equivalente en centígrados del valor en Fahrenheit   *Galones a litros:*  Devolver el equivalente en litros del valor en galones   *Programa principal* (main())  Fundamentos de la programación: La abstracción procedimental *Página 484*  Luis Hernández Yáñez  Paso 3.‐ Cada tarea, un subprograma  Comunicación entre los subprogramas:  Fundamentos de la programación: La abstracción procedimental *Página 485*  Función Entrada Salida Valor devuelto  menu() ― ― int  pulgACm() double pulg ― double  lbAGr() double libras ― double  grFAGrC() double grF ― double  galALtr() double galones ― double  main() ― ― int  Luis Hernández Yáñez  Paso 4.‐ Algoritmos detallados de cada subprograma  Programar  Fundamentos de la programación: La abstracción procedimental *Página 486*  . . .  #include <iostream>  using namespace std;  // Prototipos  int menu();  double pulgACm(double pulg);  double lbAGr(double libras);  double grFAGrC(double grF);  double galALtr(double galones);  int main() {  double valor;  int op = ‐1;  while (op != 0) {  op = menu();  switch (op) {  case 1:  {  cout << "Pulgadas: ";  cin >> valor;  cout << "Son " << pulgACm(valor) << " cm." << endl;  }  break;  Luis Hernández Yáñez  Fundamentos de la programación: La abstracción procedimental *Página 487*  . . .  endl;  }  break;  }  case 2:  {  cout << "Libras: ";  cin >> valor;  cout << "Son " << lbAGr(valor) << " gr." << endl;  }  break;  case 3:  {  cout << "Grados Fahrenheit: ";  cin >> valor;  cout << "Son " << grFAGrC(valor) << " ºC" << endl;  }  break;  case 4:  {  cout << "Galones: ";  cin >> valor;  cout << "Son " << galALtr(valor) << " l." << }  return 0;  }  Luis Hernández Yáñez  Fundamentos de la programación: La abstracción procedimental *Página 488*  int menu() {  int op = ‐1;  while ((op < 0) || (op > 4)) {  cout << "1 ‐ Pulgadas a Cm." << endl;  cout << "2 ‐ Libras a Gr." << endl;  cout << "3 ‐ Fahrenheit a ºC" << endl;  0) || (op > 4)) {  cout << "Opción no válida" << endl;  }  }  return op;  }  double pulgACm(double pulg) {  const double cmPorPulg = 2.54;  return pulg \* cmPorPulg;  }  cout << "4 ‐ Galones a L." << endl;  cout << "0 ‐ Salir" << endl;  cout << "Elige: ";  cin >> op;  if ((op < . . .  Luis Hernández Yáñez  Fundamentos de la programación: La abstracción procedimental *Página 489*  double lbAGr(double libras) {  const double grPorLb = 453.6;  return libras \* grPorLb;  }  double grFAGrC(double grF) {  return ((grF ‐ 32) \* 5 / 9);  }  double galALtr(double galones) {  const double ltrPorGal = 4.54609;  return galones \* ltrPorGal;  }  . . .  conversiones.cpp  Luis Hernández Yáñez  Fundamentos de la programación: La abstracción procedimental Página 490  Luis Hernández Yáñez  *Integridad de los subprogramas*  Condiciones que se deben dar antes de comenzar su ejecución   Precondiciones   Quien llame al subprograma debe garantizar que se satisfacen  Condiciones que se darán cuando termine su ejecución   Postcondiciones   En el punto de llamada se pueden dar por garantizadas  *Aserciones*:  Condiciones que si no se cumplen interrumpen la ejecución  Función assert()  Fundamentos de la programación: La abstracción procedimental Página 491  Luis Hernández Yáñez  *Precondiciones*  Por ejemplo, no realizaremos conversiones de valores negativos:  double pulgACm(double pulg) {  assert(pulg > 0);  double cmPorPulg = 2.54;  return pulg \* cmPorPulg;  }  La función tiene una precondición: pulg debe ser positivo  assert(pulg > 0); interrumpirá la ejecución si no es cierto  Fundamentos de la programación: La abstracción procedimental Página 492  Luis Hernández Yáñez  *Precondiciones*  Es responsabilidad del punto de llamada garantizar la precondición:  int main() {  double valor;  int op = ‐1;  while (op != 0) {  op = menu();  switch (op) {  case 1:  {  cout << "Pulgadas: ";  cin >> valor;  if (valor < 0) {  cout << "¡No válido!" << endl;  }  else { // Se cumple la precondición...  ...  Fundamentos de la programación: La abstracción procedimental Página 493  Luis Hernández Yáñez  *Postcondiciones*  Un subprograma puede garantizar condiciones al terminar:  int menu() {  int op = ‐1;  while ((op < 0) || (op > 4)) {  ...  cout << "Elige: ";  cin >> op;  if ((op < 0) || (op > 4)) {  cout << "Opción no válida" << endl;  }  }  assert ((op >= 0) && (op <= 4));  return op;  }  El subprograma debe asegurarse de que se cumpla  Fundamentos de la programación: La abstracción procedimental Página 494  Luis Hernández Yáñez  *Licencia CC (Creative Commons)*  Este tipo de licencias ofrecen algunos derechos a terceras personas  bajo ciertas condiciones.  Este documento tiene establecidas las siguientes:  Pulsa en la imagen de arriba a la derecha para saber más.  Fundamentos de la programación: La abstracción procedimental Página 495  Reconocimiento (*Attribution*):  En cualquier explotación de la obra autorizada por la licencia  hará falta reconocer la autoría.  No comercial (*Non commercial*):  La explotación de la obra queda limitada a usos no comerciales.  Compartir igual (*Share alike*):  La explotación autorizada incluye la creación de obras derivadas  siempre que mantengan la misma licencia al ser divulgadas.  **4A**  Grado en Ingeniería Informática  Grado en Ingeniería del Software  Grado en Ingeniería de Computadores  Luis Hernández Yáñez  Facultad de Informática  Universidad Complutense  Fundamentos de la programación  Luis Hernández Yáñez  Fundamentos de la programación: La abstracción procedimental (Anexo)  Archivos como parámetros 498  La función main() 501  Argumentos implícitos 504  Sobrecarga de subprogramas 508  Luis Hernández Yáñez  Fundamentos de la programación: La abstracción procedimental (Anexo) Página 498  Luis Hernández Yáñez  #include <iostream>  using namespace std;  #include <fstream>  void sumatorio\_archivo(ifstream &arch, double &suma);  int main() {  double resultado;  ifstream archivo;  archivo.open("datos.txt");  if (!archivo.is\_open()) {  cout << "ERROR DE APERTURA" << endl;  }  else {  sumatorio\_archivo(archivo, resultado)  cout << "Suma = " << resultado << endl;  archivo.close();  }  return 0;  }  Fundamentos de la programación: La abstracción procedimental (Anexo) Página 499  Luis Hernández Yáñez  void sumatorio\_archivo(ifstream &arch, double &suma) {  double dato;  suma = 0;  arch >> dato;  while (dato != ‐1) {  suma = suma + dato;  arch >> dato;  }  }  Fundamentos de la programación: La abstracción procedimental (Anexo) Página 500  Los archivos siempre se pasan por referencia  Luis Hernández Yáñez  Fundamentos de la programación: La abstracción procedimental (Anexo) Página 501  Luis Hernández Yáñez  *Comunicación con el sistema operativo*  Parámetros opcionales de la función main():  int main(int argc, char \*argv[])  Para obtener datos proporcionados al ejecutar el programa:  C:\>prueba *cad1 cad2 cad3*  Ejecuta prueba.exe con tres argumentos (cadenas)  Parámetros de main():  — argc: número de argumentos que se proporcionan  4 en el ejemplo (primero: nombre del programa con su ruta)  — argv: array con las cadenas proporcionadas como argumentos  Fundamentos de la programación: La abstracción procedimental (Anexo) Página 502  Luis Hernández Yáñez  *¿Cómo ha ido la función?*  La función main() devuelve al S.O. un código de terminación  — 0: *Todo OK*  — Distinto de 0: *¡Ha habido un error!*  Si la ejecución llega al final de la función main(), todo OK:  ...  return 0; // Fin del programa  }  Fundamentos de la programación: La abstracción procedimental (Anexo) Página 503  Luis Hernández Yáñez  Fundamentos de la programación: La abstracción procedimental (Anexo) Página 504  Luis Hernández Yáñez  *Valores predeterminados para parámetros por valor*  Valor por defecto para un parámetro:  Tras un = a continuación del nombre del parámetro:  void proc(int i = 1);  Si no se proporciona argumento, el parámetro toma ese valor  proc(12); i toma el valor explícito 12  proc(); i toma el valor implícito (1)  Sólo puede haber argumentos implícitos en los parámetros finales:  void p(int i, int j = 2, int k = 3); // CORRECTO  void p(int i = 1, int j, int k = 3); // INCORRECTO  Fundamentos de la programación: La abstracción procedimental (Anexo) Página 505  Una vez asignado un valor implícito, todos los que siguen  han de tener también valor implícito  Luis Hernández Yáñez  *Parámetros y argumentos implícitos*  void p(int i, int j = 2, int k = 3);  Se copian los argumentos en los parámetros del primero al último   los que no tengan correspondencia tomarán los implícitos  void p(int i, int j = 2, int k = 3);  ...  p(13); // i toma 13, j y k sus valores implícitos  p(5, 7); // i toma 5, j toma 7 y k su valor implícito  p(3, 9, 12); // i toma 3, j toma 9 y k toma 12  Fundamentos de la programación: La abstracción procedimental (Anexo) Página 506  Los argumentos implícitos se declaran en el prototipo  (preferible) o en la cabecera del subprograma, pero NO en ambos  Luis Hernández Yáñez  #include <iostream>  using namespace std;  double f(double x, double y, int signo = 1, double delta = 1.0);  int main() {  double x, y;  cout << "X = ";  cin >> x;  cout << "Y = ";  cin >> y;  cout << "signo y delta por defecto: " << f(x, y) << endl;  cout << "signo ‐1 y delta por defecto: " << f(x, y, ‐1) << endl;  cout << "signo y delta concretos: " << f(x, y, 1, 1.25) << endl;  return 0;  }  double f(double x, double y, int signo, double delta) {  return signo \* delta \* x / y;  }  Fundamentos de la programación: La abstracción procedimental (Anexo) Página 507  Por defecto, signo +  Por defecto, Δ es 1  No podemos dejar signo por defecto  y concretar delta  Luis Hernández Yáñez  Fundamentos de la programación: La abstracción procedimental (Anexo) Página 508  Luis Hernández Yáñez  *Igual nombre, distintos parámetros*  Funciones o procedimientos con igual nombre y distintos parámetros:  int abs(int n);  double abs(double n);  long int abs(long int n);  Se ejecutará la función que corresponda al tipo de argumento:  abs(13) // argumento int ‐‐> primera función  abs(‐2.3) // argumento double ‐‐> segunda función  abs(3L) // argumento long int ‐‐> tercera función  Fundamentos de la programación: La abstracción procedimental (Anexo) Página 509  Para indicar que es un literal long int, en lugar de int  Luis Hernández Yáñez  #include <iostream>  using namespace std;  void intercambia(int &x, int &y);  void intercambia(double &x,  double &y);  void intercambia(char &x, char &y);  void intercambia(int &x, int &y) {  int tmp;  tmp = x;  x = y;  y = tmp;  }  void intercambia(double &x,  double &y) {  double tmp;  tmp = x;  x = y;  y = tmp;  }  void intercambia(char &x, char &y) {  char tmp;  tmp = x;  x = y;  y = tmp;  }  int main() {  int i1 = 3, i2 = 7;  double d1 = 12.5, d2 = 35.9;  char c1 = 'a', c2 = 'b';  cout << i1 << " ‐ " << i2 << endl;  cout << d1 << " ‐ " << d2 << endl;  cout << c1 << " ‐ " << c2 << endl;  intercambia(i1, i2);  intercambia(d1, d2);  intercambia(c1, c2);  cout << i1 << " ‐ " << i2 << endl;  cout << d1 << " ‐ " << d2 << endl;  cout << c1 << " ‐ " << c2 << endl;  return 0;  }  Fundamentos de la programación: La abstracción procedimental (Anexo) Página 510  inter.cpp  Luis Hernández Yáñez  *Licencia CC (Creative Commons)*  Este tipo de licencias ofrecen algunos derechos a terceras personas  bajo ciertas condiciones.  Este documento tiene establecidas las siguientes:  Pulsa en la imagen de arriba a la derecha para saber más.  Fundamentos de la programación: La abstracción procedimental (Anexo) Página 511  Reconocimiento (*Attribution*):  En cualquier explotación de la obra autorizada por la licencia  hará falta reconocer la autoría.  No comercial (*Non commercial*):  La explotación de la obra queda limitada a usos no comerciales.  Compartir igual (*Share alike*):  La explotación autorizada incluye la creación de obras derivadas  siempre que mantengan la misma licencia al ser divulgadas.  **5**  Grado en Ingeniería Informática  Grado en Ingeniería del Software  Grado en Ingeniería de Computadores  Luis Hernández Yáñez/Pablo Moreno Ger  Facultad de Informática  Universidad Complutense  Fundamentos de la programación  Luis Hernández Yáñez/Pablo Moreno Ger  Fundamentos de la programación: Tipos de datos estructurados  Tipos de datos 514  Arrays de nuevo 517  Arrays y bucles for 520  Más sobre arrays 522  Inicialización de arrays 523  Enumerados como índices 524  Paso de arrays a subprogramas 525  Implementación de listas 528  Cadenas de caracteres 531  Cadenas de caracteres de tipo string 535  Entrada/salida con string 539  Operaciones con string 541  Estructuras 543  Estructuras dentro de estructuras 549  Arrays de estructuras 550  Arrays dentro de estructuras 551  Listas de longitud variable 552  Un ejemplo completo 558  El bucle do..while 562  Luis Hernández Yáñez/Pablo Moreno Ger  Fundamentos de la programación: Tipos de datos estructurados Página 514  Luis Hernández Yáñez/Pablo Moreno Ger  *Clasificación de tipos*   Simples   Estándar: int, float, double, char, bool  Conjunto de valores predeterminado   Definidos por el usuario: *enumerados*  Conjunto de valores definido por el programador   Estructurados   Colecciones homogéneas: *arrays*  Todos los elementos del mismo tipo   Colecciones heterogéneas: *estructuras*  Los elementos pueden ser de tipos distintos  Fundamentos de la programación: Tipos de datos estructurados Página 515        Luis Hernández Yáñez/Pablo Moreno Ger  *Colecciones o tipos aglomerados*  Agrupaciones de datos (elementos):   Todos del mismo tipo: *array* o *tabla*   De tipos distintos: *estructura*, *registro* o *tupla*  Arrays (tablas)   Elementos organizados por posición: 0, 1, 2, 3, ...   Acceso por índice: 0, 1, 2, 3, ...   Una o varias dimensiones  Estructuras (tuplas, registros)   Elementos (campos) sin orden establecido   Acceso por nombre  Fundamentos de la programación: Tipos de datos estructurados Página 516  Luis Hernández Yáñez/Pablo Moreno Ger  Fundamentos de la programación: Tipos de datos estructurados Página 517  Luis Hernández Yáñez/Pablo Moreno Ger  Fundamentos de la programación: Tipos de datos estructurados Página 518  []  *Estructura secuencial*  Cada elemento se encuentra en una posición (*índice*):   Los índices son enteros positivos   El índice del primer elemento siempre es 0   Los índices se incrementan de uno en uno  *Acceso directo*  A cada elemento se accede a través de su índice:  ventas[4] accede al 5º elemento (contiene el valor 435.00)  cout << ventas[4];  ventas[4] = 442.75;  ventas 125.40 76.95 328.80 254.62 435.00 164.29 0.00  0 1 2 3 4 5 6  Datos de un mismo tipo base:  Se usan como cualquier variable  Luis Hernández Yáñez/Pablo Moreno Ger  *Declaración de tipos de arrays*  const int *Dimensión* = ...;  typedef *tipo\_base* t*Nombre*[*Dimensión*];  Ejemplo:  const int Dias = 7;  typedef double tVentas[Dias];  Declaración de variables de tipos array: como cualquier otra  tVentas ventas;  ¡NO se inicializan los elementos automáticamente!  ¡Es responsabilidad del programador usar índices válidos!  No se pueden copiar arrays directamente (array1 = array2)  Hay que copiarlos elemento a elemento  Fundamentos de la programación: Tipos de datos estructurados Página 519  Luis Hernández Yáñez/Pablo Moreno Ger  *Procesamiento de arrays*   Recorridos   Búsquedas   Ordenación etcétera...  *Recorrido de arrays con bucles for*  Arrays: tamaño fijo  Bucles de recorrido fijo (for)  tVentas ventas;  double media, total = 0;  ...  for (int i = 0; i < Dias; i++) {  total = total + ventas[i];  }  media = total / Dias;  Fundamentos de la programación: Tipos de datos estructurados Página 520  const int Dias = 7;  typedef double tVentas[Dias];  Luis Hernández Yáñez/Pablo Moreno Ger  tVentas ventas;  double media, total = 0;  ...  for (int i = 0; i < Dias; i++) {  total = total + ventas[i];  }  Memoria  Dias **7**  ventas[0] **12.40**  ventas[1] **10.96**  ventas[2] **8.43**  ventas[3] **11.65**  ventas[4] **13.70**  ventas[5] **13.41**  ventas[6] **14.07**  media **?**  total **0.00**  i **0**  **12.40**  **1**  **23.36**  **2**  **31.79**  **3**  **43.44**  **4**  **84.62**  **7**  Fundamentos de la programación: Tipos de datos estructurados Página 521  12.40 10.96 8.43 11.65 13.70 13.41 14.07  0 1 2 3 4 5 6  true false  total+=ventas[i]  i++  i<Dias  i = 0  ...  Luis Hernández Yáñez/Pablo Moreno Ger  Fundamentos de la programación: Tipos de datos estructurados Página 522  Luis Hernández Yáñez/Pablo Moreno Ger  Podemos inicializar los elementos de los arrays en la declaración  *Asignamos* una serie de valores al array:  const int DIM = 10;  typedef int tTabla[DIM];  tTabla i = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };  Se asignan los valores por su orden:  i[0] i[1] i[2] i[3] i[4] ... i[9]  1º 2º 3º 4º 5º ... 10º  Si hay menos valores que elementos, los restantes se ponen a 0  tTabla i = { 0 }; // Pone todos los elementos a 0  Fundamentos de la programación: Tipos de datos estructurados Página 523  Luis Hernández Yáñez/Pablo Moreno Ger  const int Colores = 3,  typedef enum { rojo, verde, azul } tRGB;  typedef int tColor[Colores];  tColor color;  ...  cout << "Cantidad de rojo (0‐255): ";  cin >> color[rojo];  cout << "Cantidad de verde (0‐255): ";  cin >> color[verde];  cout << "Cantidad de azul (0‐255): ";  cin >> color[azul];  Fundamentos de la programación: Tipos de datos estructurados Página 524  Recuerda que internamente se asignan enteros a partir de 0  a los distintos símbolos del enumerado  rojo  0 verde  1 azul  2  Luis Hernández Yáñez/Pablo Moreno Ger  *Simulación de paso de parámetro por referencia*  Sin poner & en la declaración del parámetro  Los subprogramas reciben la dirección en memoria del array  const int Max = 10;  typedef int tTabla[Max];  void inicializa(tTabla tabla); // Sin poner &  Las modificaciones del array quedan reflejadas en el argumento  inicializa(array);  Si inicializa() modifica algún elemento de tabla,  automáticamente queda modificado ese elemento de array  *¡Son el mismo array!*  Fundamentos de la programación: Tipos de datos estructurados Página 525  Luis Hernández Yáñez/Pablo Moreno Ger  Fundamentos de la programación: Tipos de datos estructurados Página 526  0 1 2 3 4 5 6 7 8 9  const int Dim = 10;  typedef int tTabla[Dim];  void inicializa(tTabla tabla); // no se usa &  void inicializa(tTabla tabla) {  for (int i = 0; i < Dim; i++)  tabla[i] = i;  }  int main() {  tTabla array;  inicializa(array); // array queda modificado  for (int i = 0; i < Dim; i++)  cout << array[i] << " ";  ...  Luis Hernández Yáñez/Pablo Moreno Ger  *¿Cómo evitar que se modifique el array?*  Usando el modificador const en la declaración del parámetro:  const tTabla tabla Un array de constantes  void muestra(const tTabla tabla);  El argumento se tratará como un array de constantes  Si en el subprograma hay alguna instrucción que intente  modificar un elemento del array: error de compilación  void muestra(const tTabla tabla) {  for (int i = 0; i < Dim; i++) {  cout << tabla[i] << " ";  // OK. Se accede, pero no se modifica  }  }  Fundamentos de la programación: Tipos de datos estructurados Página 527  Luis Hernández Yáñez/Pablo Moreno Ger  Fundamentos de la programación: Tipos de datos estructurados Página 528  Luis Hernández Yáñez/Pablo Moreno Ger  *Listas con un número fijo de elementos*  Array con el nº de elementos como dimensión  const int NUM = 100;  typedef double tLista[NUM]; // Exactamente 100 double  tLista lista;  Recorrido de la lista:  for (int i = 0; i < NUM; i++) {  ...  Búsqueda en la lista:  while ((i < NUM) && !encontrado) {  ...  Fundamentos de la programación: Tipos de datos estructurados Página 529  Luis Hernández Yáñez/Pablo Moreno Ger  *Listas con un número variable de elementos*  Array con un máximo de elementos + Contador de elementos  const int MAX = 100;  typedef double tLista[MAX]; // Hasta 100 elementos  tLista lista;  int contador = 0; // Se incrementa al insertar  Recorrido de la lista:  for (int i = 0; i < contador; i++) {  ...  Búsqueda en la lista:  while ((i < contador) && !encontrado) {  ...  Fundamentos de la programación: Tipos de datos estructurados Página 530  ¿Array y contador por separado?  Estructuras  Luis Hernández Yáñez/Pablo Moreno Ger  Fundamentos de la programación: Tipos de datos estructurados Página 531  Luis Hernández Yáñez/Pablo Moreno Ger  *Arrays de caracteres*  Cadenas: secuencias de caracteres de longitud variable  "Hola" "Adiós" "Supercalifragilístico" "1234 56 7"  Variables de cadena: contienen secuencias de caracteres  Se guardan en arrays de caracteres: tamaño máximo (dimensión)  No todas las posiciones del array son relevantes:   Longitud de la cadena: número de caracteres, desde el  primero, que realmente constituyen la cadena:  Longitud actual: 4  Fundamentos de la programación: Tipos de datos estructurados Página 532  H o l a  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21  Luis Hernández Yáñez/Pablo Moreno Ger  *Longitud de la cadena*  Longitud: 5  Longitud: 21  Necesidad de saber dónde terminan los caracteres relevantes:   Mantener la longitud de la cadena como dato asociado   Colocar un carácter de terminación al final (*centinela*)  Fundamentos de la programación: Tipos de datos estructurados Página 533  A d i ó s  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21  S u p e r c a l i f r a g i l í s t i c o  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21  A d i ó s \0  0 1 2 3 4 5 6 7 8 9 10  Luis Hernández Yáñez/Pablo Moreno Ger  *Cadenas de caracteres en C++*  Dos alternativas para el manejo de cadenas:   Cadenas al estilo de C (*terminadas en nulo*)   Tipo string  Cadenas al estilo de C *Anexo del tema*   Arrays de tipo char con una longitud máxima   Un último carácter especial al final: '\0'  Tipo string   Cadenas más sofisticadas   Sin longitud máxima (gestión automática de la memoria)   Multitud de funciones de utilidad (biblioteca string)  Fundamentos de la programación: Tipos de datos estructurados Página 534  Luis Hernández Yáñez/Pablo Moreno Ger  Fundamentos de la programación: Tipos de datos estructurados Página 535  Luis Hernández Yáñez/Pablo Moreno Ger  *El tipo* string   El tipo asume la responsabilidad de la gestión de memoria   Define operadores sobrecargados (+ para concatenar)   Cadenas más eficientes y seguras de usar  Biblioteca string  Requiere establecer el espacio de nombres a std   Se pueden inicializar en la declaración   Se pueden copiar con el operador de asignación   Se pueden concatenar con el operador +   Multitud de funciones de utilidad  Fundamentos de la programación: Tipos de datos estructurados Página 536  Luis Hernández Yáñez/Pablo Moreno Ger  #include <iostream>  #include <string>  using namespace std;  int main() {  string cad1("Hola"); // inicialización  string cad2 = "amigo"; // inicialización  string cad3;  cad3 = cad1; // copia  cout << "cad3 = " << cad3 << endl;  cad3 = cad1 + " "; // concatenación  cad3 += cad2; // concatenación  cout << "cad3 = " << cad3 << endl;  cad1.swap(cad2); // intercambio  cout << "cad1 = " << cad1 << endl;  cout << "cad2 = " << cad2 << endl;  return 0;  }  Fundamentos de la programación: Tipos de datos estructurados Página 537  string.cpp  Luis Hernández Yáñez/Pablo Moreno Ger  Longitud de la cadena:  cadena.length() o cadena.size()  Se pueden comparar con los operadores relacionales:  if (cad1 <= cad2) { ...  Acceso a los caracteres de una cadena:   Como array de caracteres: *cadena*[*i*]  Sin control de acceso a posiciones inexistentes del array  Sólo debe usarse si se está seguro de que el índice es válido   Función at(*índice*): *cadena*.at(*i*)  Error de ejecución si se accede a una posición inexistente  Fundamentos de la programación: Tipos de datos estructurados Página 538  Luis Hernández Yáñez/Pablo Moreno Ger   Se muestran en la pantalla con cout <<   Lectura con cin >>: termina con espacio en blanco (inc. Intro)  El espacio en blanco queda pendiente   Descartar el resto de los caracteres del búfer:  cin.sync();   Lectura incluyendo espacios en blanco:  getline(cin, *cadena*)  Guarda en la *cadena* los caracteres leídos hasta el fin de línea   Lectura de archivos de texto:  Igual que de consola; sync() no tiene efecto  archivo >> cadena getline(archivo, cadena)  Fundamentos de la programación: Tipos de datos estructurados Página 539  Luis Hernández Yáñez/Pablo Moreno Ger  Fundamentos de la programación: Tipos de datos estructurados Página 540  string2.cpp  #include <iostream>  #include <string>  using namespace std;  int main() {  string nombre, apellidos;  cout << "Introduzca un nombre: ";  cin >> nombre;  cout << "Introduzca los apellidos: ";  cin.sync();  getline(cin, apellidos);  cout << "Nombre completo: " << nombre << " "  << apellidos << endl;  return 0;  }  Luis Hernández Yáñez/Pablo Moreno Ger   *cadena*.substr(*posición*, *longitud*)  Subcadena de *longitud* caracteres desde *posición*  string cad = "abcdefg";  cout << cad.substr(2, 3); // Muestra cde   *cadena*.find(*subcadena*)  Posición de la primera ocurrencia de *subcadena* en *cadena*  string cad = "Olala";  cout << cad.find("la"); // Muestra 1  (Recuerda que los arrays de caracteres comienzan con el índice 0)   *cadena*.rfind(*subcadena*)  Posición de la última ocurrencia de *subcadena* en *cadena*  string cad = "Olala";  cout << cad.rfind("la"); // Muestra 3  Fundamentos de la programación: Tipos de datos estructurados Página 541  Luis Hernández Yáñez/Pablo Moreno Ger   *cadena*.erase(*ini*, *num*)  Elimina *num* caracteres a partir de la posición *ini*  string cad = "abcdefgh";  cad.erase(3, 4); // cad ahora contiene "abch"   *cadena*.insert(*ini*, *cadena2*)  Inserta *cadena2* a partir de la posición *ini*  string cad = "abcdefgh";  cad.insert(3, "123"); // cad ahora contiene "abc123defgh"  http://www.cplusplus.com/reference/string/string/  Fundamentos de la programación: Tipos de datos estructurados Página 542  Luis Hernández Yáñez/Pablo Moreno Ger  Fundamentos de la programación: Tipos de datos estructurados Página 543  Luis Hernández Yáñez/Pablo Moreno Ger  *Colecciones heterogéneas (tuplas, registros)*  Elementos de (posiblemente) distintos tipos: *campos*  Campos identificados por su nombre  *Información relacionada que se puede manejar como una unidad*  Acceso a cada elemento por su nombre de campo (operador.)  Fundamentos de la programación: Tipos de datos estructurados Página 544  Luis Hernández Yáñez/Pablo Moreno Ger  typedef struct {  ... // declaraciones de campos (como variables)  } *tTipo*; // nombre de tipo ‐ ¡al final!  typedef struct {  string nombre;  string apellidos;  int edad;  string nif;  } tPersona;  Campos:  Tipos estándar o previamente declarado  Fundamentos de la programación: Tipos de datos estructurados Página 545  Luis Hernández Yáñez/Pablo Moreno Ger  tPersona persona;  Las variables de tipo tPersona contienen cuatro datos (campos):  nombre apellidos edad nif  Acceso a los campos con el operador punto (.):  persona.nombre // una cadena (string)  persona.apellidos // una cadena (string)  persona.edad // un entero (int)  persona.nif // una cadena (string)  Podemos copiar dos estructuras directamente:  tPersona persona1, persona2;  ...  persona2 = persona1;  Se copian todos los campos a la vez  Fundamentos de la programación: Tipos de datos estructurados Página 546  Luis Hernández Yáñez/Pablo Moreno Ger  typedef struct {  string nombre;  string apellidos;  int edad;  string nif;  } tPersona;  tPersona persona;  Fundamentos de la programación: Tipos de datos estructurados Página 547  nombre Luis Antonio  apellidos Hernández Yáñez  edad 22  nif 00223344F  Memoria  persona.nombre **Luis**  **Antonio**  persona.apellidos **Hernández**  **Yáñez**  persona.edad **22**  persona.nif **00223344F**  persona  Luis Hernández Yáñez/Pablo Moreno Ger  typedef struct {  string nombre;  string apellidos;  int edad;  string nif;  } tPersona;  tPersona persona;  Los campos no siguen ningún orden establecido  Acceso directo por nombre de campo (operador .)  Con cada campo se puede hacer lo que permita su tipo  Fundamentos de la programación: Tipos de datos estructurados Página 548  Las estructuras se pasan por valor (sin &)  o por referencia (con &) a los subprogramas  Luis Hernández Yáñez/Pablo Moreno Ger  typedef struct { typedef struct {  string dni; ...  char letra; tNif nif;  } tNif; } tPersona;  tPersona persona;  Acceso al NIF completo:  persona.nif // Otra estructura  Acceso a la letra del NIF:  persona.nif.letra  Acceso al DNI:  persona.nif.dni  Fundamentos de la programación: Tipos de datos estructurados Página 549  nombre  apellidos  edad  nif  tPersona  dni  letra  tNif  Luis Hernández Yáñez/Pablo Moreno Ger  const int DIM = 100;  typedef struct {  string nombre;  string apellidos;  int edad;  string nif;  } tPersona;  typedef tPersona tArray[DIM];  tArray personal;  Nombre de la tercera persona:  personal[2].nombre  Edad de la duodécima persona:  personal[11].edad  NIF de la primera persona:  personal[0].nif  Fundamentos de la programación: Tipos de datos estructurados Página 550  nombre  apellidos  edad  nif  tPersona  nombre  apellidos  edad  nif  nombre  apellidos  edad  nif  nombre  apellidos  edad  nif  nombre  apellidos  edad  nif  1  0  2  DIM‐1  personal  Luis Hernández Yáñez/Pablo Moreno Ger  const int MAX = 100;  typedef struct {  string nombre;  string apellidos;  int edad;  string nif;  } tPersona;  typedef tPersona tArray[MAX];  typedef struct {  tArray elementos;  int contador;  } tLista;  tLista lista;  Nombre de la tercera persona: lista.elementos[2].nombre  Edad de la duodécima persona: lista.elementos[11].edad  NIF de la primera persona: lista.elementos[0].nif  Fundamentos de la programación: Tipos de datos estructurados Página 551  nombre  apellidos  edad  nif  nombre  apellidos  edad  nif  nombre  apellidos  edad  nif  nombre  apellidos  edad  nif  1  0  2  MAX‐1  elementos  lista contador  Luis Hernández Yáñez/Pablo Moreno Ger  Fundamentos de la programación: Tipos de datos estructurados Página 552  Luis Hernández Yáñez/Pablo Moreno Ger / Pablo Moreno Ger  Estructura que agrupe el array y el contador:  const int MAX = 10;  typedef double tArray[MAX];  typedef struct {  tArray elementos;  int contador;  } tLista;  Operaciones principales: inserción y eliminación de elementos  contador 6  Fundamentos de la programación: Tipos de datos estructurados Página 553  elementos  12.0 ‐2.2 5.4 0.0 36.2 35.0 X X X X  0 1 2 3 4 5 6 7 8 9  Nº de elementos (y primer índice sin elemento)  Elementos sin usar  (datos basura)  Luis Hernández Yáñez/Pablo Moreno Ger  *Insertar un nuevo elemento en una posición*  Posiciones válidas: 0 a contador  Hay que asegurarse de que haya sitio (contador < máximo)  Operación en 3 pasos:  1.‐ Abrir hueco para el nuevo elemento (desde la posición)  2.‐ Colocar el elemento nuevo en la posición  3.‐ Incrementar el contador  6  12.0 ‐2.2 5.4 0.0 36.2 35.0 X X X X  0 1 2 3 4 5 6 7 8 9  42.0  nuevo  Fundamentos de la programación: Tipos de datos estructurados Página 554  3  pos  Luis Hernández Yáñez/Pablo Moreno Ger  if (lista.contador < N) {  // Abrir hueco  for (int i = lista.contador; i > pos; i‐‐) {  lista.elementos[i] = lista.elementos[i ‐ 1];  }  // Insertar e incrementar contador  lista.elementos[pos] = nuevoElemento;  lista.contador++;  }  Fundamentos de la programación: Tipos de datos estructurados Página 555  7  12.0 ‐2.2 5.4 42.0 0.0 36.2 35.0 X X X  0 1 2 3 4 5 6 7 8 9  42.0  nuevo  3  pos  Luis Hernández Yáñez/Pablo Moreno Ger  *Eliminar el elemento en una posición*  Posiciones válidas: 0 a contador‐1  Desplazar a la izquierda desde el siguiente y decrementar el contador:  for (int i = pos; i < lista.contador ‐ 1 ; i++) {  lista.elementos[i] = lista.elementos[i + 1];  }  lista.contador‐‐;  6  12.0 ‐2.2 5.4 0.0 36.2 35.0 X X X X  0 1 2 3 4 5 6 7 8 9  Fundamentos de la programación: Tipos de datos estructurados Página 556  3  pos  Luis Hernández Yáñez/Pablo Moreno Ger  5  3  pos  for (int i = pos; i < lista.contador ‐ 1 ; i++) {  lista.elementos[i] = lista.elementos[i + 1];  }  lista.contador‐‐;  6  12.0 ‐2.2 5.4 0.0 36.2 35.0 X X X X  0 1 2 3 4 5 6 7 8 9  Fundamentos de la programación: Tipos de datos estructurados Página 557  3  pos  12.0 ‐2.2 5.4 36.2 35.0 35.0 X X X X  0 1 2 3 4 5 6 7 8 9  Luis Hernández Yáñez/Pablo Moreno Ger  Fundamentos de la programación: Tipos de datos estructurados Página 558  Luis Hernández Yáñez/Pablo Moreno Ger  Fundamentos de la programación: Tipos de datos estructurados Página 559  *Descripción*  Programa que mantenga una lista de los estudiantes de una clase  De cada estudiante: nombre, apellidos, edad, NIF y nota   Se desconoce el número total de estudiantes (máximo 100)   La información de la lista se mantiene en un archivo clase.txt  Se carga al empezar y se guarda al finalizar   El programa debe ofrecer estas opciones:  — Añadir un nuevo alumno  — Eliminar un alumno existente  — Calificar a los estudiantes  — Listado de notas, identificando la mayor y la media  Luis Hernández Yáñez/Pablo Moreno Ger  #include <iostream>  #include <string>  using namespace std;  #include <fstream>  #include <iomanip>  const int MAX = 100;  typedef struct {  string nombre;  string apellidos;  int edad;  string nif;  double nota;  } tEstudiante;  typedef tEstudiante tArray[MAX];  typedef struct {  tArray elementos;  int contador;  } tLista;  Fundamentos de la programación: Tipos de datos estructurados Página 560  bd.cpp  Declaraciones de constantes  y tipos globales  Tras las bibliotecas  Luis Hernández Yáñez/Pablo Moreno Ger  // Prototipos  int menu(); // Menú del programa ‐ devuelve la opción elegida  void cargar(tLista &lista, bool &ok); // Carga del archivo  void guardar(const tLista &lista); // La guarda en el archivo  void leerEstudiante(tEstudiante &estudiante); // Lee los datos  void insertarEstudiante(tLista &lista, tEstudiante estudiante,  bool &ok); // Inserta un nuevo estudiante en la lista  void eliminarEstudiante(tLista &lista, int pos, bool &ok);  // Elimina el estudiante en esa posición  string nombreCompleto(tEstudiante estudiante);  void calificar(tLista &lista); // Notas de los estudiantes  double mediaClase(const tLista &lista); // Nota media  int mayorNota(const tLista &lista);  // Índice del estudiante con mayor nota  void mostrarEstudiante(tEstudiante estudiante);  void listado(const tLista &lista, double media, int mayor);  // Listado de la clase  Fundamentos de la programación: Tipos de datos estructurados Página 561  Los prototipos, después de los tipos globales  Luis Hernández Yáñez/Pablo Moreno Ger  Fundamentos de la programación: Tipos de datos estructurados Página 562  Luis Hernández Yáñez/Pablo Moreno Ger  *El bucle* do..while  do *cuerpo* while (*condición*);  int i = 1;  do {  cout << i << endl;  i++;  } while (i <= 100);  El *cuerpo* siempre se ejecuta al menos una vez  El *cuerpo* es un bloque de código  Fundamentos de la programación: Tipos de datos estructurados Página 563  Condición al final del bucle  do *cuerpo* while ( *condición* ) ;  Luis Hernández Yáñez/Pablo Moreno Ger  int i = 1;  do {  cout << i << endl;  i++;  } while (i <= 100);  Fundamentos de la programación: Tipos de datos estructurados Página 564  true  false  *Condición*  *Cuerpo*  false  true  i <= 100  cout << i << endl;  i++;  i = 1;  El cuerpo  se ejecuta  al menos  una vez  Luis Hernández Yáñez/Pablo Moreno Ger  *¿Ha de ejecutarse al menos una vez el cuerpo del bucle?*  cin >> d; // Lectura del 1º do {  while (d != 0) { cin >> d;  suma = suma + d; if (d != 0) { // ¿Final?  cont++; suma = suma + d;  cin >> d; cont++;  } }  } while (d != 0);  cout << "Opción: "; do { // Más simple  cin >> op; // Lectura del 1º cout << "Opción: ";  while ((op < 0) || (op > 4)) { cin >> op;  cout << "Opción: "; } while ((op < 0) || (op > 4));  cin >> op;  }  Fundamentos de la programación: Tipos de datos estructurados Página 565  Luis Hernández Yáñez/Pablo Moreno Ger  int menu() {  int op;  do {  cout << "1 ‐ Añadir un nuevo estudiante" << endl;  cout << "2 ‐ Eliminar un estudiante" << endl;  cout << "3 ‐ Calificar a los estudiantes" << endl;  cout << "4 ‐ Listado de estudiantes" << endl;  cout << "0 ‐ Salir" << endl;  cout << "Opción: ";  cin >> op;  } while ((op < 0) || (op > 4));  return op;  }  Fundamentos de la programación: Tipos de datos estructurados Página 566  Luis Hernández Yáñez/Pablo Moreno Ger  Fundamentos de la programación: Tipos de datos estructurados Página 567  *El archivo* clase.txt  Un dato en cada línea  Por cada estudiante:   Nombre (cadena)   Apellidos (cadena)   Edad (entero)   NIF (cadena)   Nota (real; ‐1 si no calificado)  Termina con XXX como nombre  El archivo se supone correcto  ↲  ↲  ↲  ↲  ↲  ↲  ↲  ↲  ↲  ↲  ↲  ↲  ↲  ↲  ↲  ↲↲  ↲  ↲  ↲  ↲  ↲  ↲  ↲  ↲  Luis Hernández Yáñez/Pablo Moreno Ger  Fundamentos de la programación: Tipos de datos estructurados Página 568  *Lectura de la información de un estudiante*  Nombre y apellidos:  Puede haber varias palabras  getline()  Edad  extractor (>>)  NIF: Una sola palabra  extractor (>>)  Nota  extractor (>>)  Queda pendiente de leer el Intro  Hay que saltar (leer) ese carácter con get()  Si no, en el siguiente nombre se leería una cadena vacía (Intro)  No leas directamente en la lista:  getline(archivo, lista.elementos[lista.contador].nombre);  Lee en una variable auxiliar de tipo tEstudiante  Luis Hernández Yáñez/Pablo Moreno Ger  Fundamentos de la programación: Tipos de datos estructurados Página 569  void cargar(tLista &lista, bool &ok) {  tEstudiante estudiante; // Variable auxiliar para leer  ifstream archivo;  char aux;  lista.contador = 0; // Inicializamos la lista  archivo.open("clase.txt");  if (!archivo.is\_open()) {  ok = false;  }  else {  ok = true;  getline(archivo, estudiante.nombre); // Leemos el primer nombre  while ((estudiante.nombre != "XXX") && (lista.contador < MAX)) {  getline(archivo, estudiante.apellidos);  archivo >> estudiante.edad;  archivo >> estudiante.nif;  archivo >> estudiante.nota;  archivo.get(aux); // Saltamos el Intro  lista.elementos[lista.contador] = estudiante; // Al final  lista.contador++;  getline(archivo, estudiante.nombre); // Siguiente nombre  } // Si hay más de MAX estudiantes, ignoramos el resto  archivo.close();  }  }  Luis Hernández Yáñez/Pablo Moreno Ger  Fundamentos de la programación: Tipos de datos estructurados Página 570  Simplemente, un dato en cada línea y en orden:  void guardar(const tLista &lista) {  ofstream archivo;  archivo.open("clase.txt");  for (int i = 0; i < lista.contador; i++) {  archivo << lista.elementos[i].nombre << endl;  archivo << lista.elementos[i].apellidos << endl;  archivo << lista.elementos[i].edad << endl;  archivo << lista.elementos[i].nif << endl;  archivo << lista.elementos[i].nota << endl;  }  archivo << "XXX" << endl; // Centinela final  archivo.close();  }  const tLista &lista  Referencia constante  Paso por referencia pero como constante  Paso por valor  Evita la copia del argumento en el parámetro (estructuras grandes)  Luis Hernández Yáñez/Pablo Moreno Ger  void leerEstudiante(tEstudiante &estudiante) {  cin.sync(); // Descartamos cualquier entrada pendiente  cout << "Nombre: ";  getline(cin, estudiante.nombre);  cout << "Apellidos: ";  getline(cin, estudiante.apellidos);  cout << "Edad: ";  cin >> estudiante.edad;  cout << "NIF: ";  cin >> estudiante.nif;  estudiante.nota = ‐1; // Sin calificar de momento  cin.sync(); // Descartamos cualquier entrada pendiente  }  Fundamentos de la programación: Tipos de datos estructurados Página 571  Luis Hernández Yáñez/Pablo Moreno Ger  void insertarEstudiante(tLista &lista, tEstudiante estudiante,  bool &ok) {  ok = true;  if (lista.contador == MAX) {  ok = false;  }  else {  lista.elementos[lista.contador] = estudiante;  // Insertamos al final  lista.contador++;  }  }  Fundamentos de la programación: Tipos de datos estructurados Página 572  Luis Hernández Yáñez/Pablo Moreno Ger  void eliminarEstudiante(tLista &lista, int pos, bool &ok) {  // Espera el índice del elemento en pos  if ((pos < 0) || (pos > lista.contador ‐ 1)) {  ok = false; // Elemento inexistente  }  else {  ok = true;  for (int i = pos; i < lista.contador ‐ 1; i++) {  lista.elementos[i] = lista.elementos[i + 1];  }  lista.contador‐‐;  }  }  Fundamentos de la programación: Tipos de datos estructurados Página 573  Luis Hernández Yáñez/Pablo Moreno Ger  string nombreCompleto(tEstudiante estudiante) {  return estudiante.nombre + " " + estudiante.apellidos;  }  void calificar(tLista &lista) {  for (int i = 0; i < lista.contador; i++) {  cout << "Nota del estudiante "  << nombreCompleto(lista.elementos[i]) << ": ";  cin >> lista.elementos[i].nota;  }  }  Fundamentos de la programación: Tipos de datos estructurados Página 574  Luis Hernández Yáñez/Pablo Moreno Ger  double mediaClase(const tLista &lista) {  double total = 0.0;  for (int i = 0; i < lista.contador; i++) {  total = total + lista.elementos[i].nota;  }  return total / lista.contador;  }  int mayorNota(const tLista &lista) {  double max = 0;  int pos = 0;  for (int i = 0; i < lista.contador; i++) {  if (lista.elementos[i].nota > max) {  max = lista.elementos[i].nota;  pos = i;  }  }  return pos;  }  Fundamentos de la programación: Tipos de datos estructurados Página 575  Luis Hernández Yáñez/Pablo Moreno Ger  Fundamentos de la programación: Tipos de datos estructurados Página 576  void mostrarEstudiante(tEstudiante estudiante) {  cout << setw(35) << left << nombreCompleto(estudiante);  cout << estudiante.nif << " ";  cout << setw(2) << estudiante.edad << " años ";  cout << fixed << setprecision(1) << estudiante.nota;  }  void listado(const tLista &lista, double media, int mayor) {  for (int i = 0; i < lista.contador; i++) {  cout << setw(3) << i << ": ";  mostrarEstudiante(lista.elementos[i]);  if (i == mayor) {  cout << " <<< Mayor nota!";  }  cout << endl;  }  cout << "Media de la clase: " << fixed << setprecision(1)  << media << endl << endl;  }  Luis Hernández Yáñez/Pablo Moreno Ger  int main() {  tLista lista;  tEstudiante estudiante;  bool exito;  int op, pos;  cargar(lista, exito);  if (!exito) {  cout << "No se ha podido cargar la lista!" << endl;  }  else {  do { // El bucle do evita tener que leer antes la primera opción  op = menu();  switch (op) {  case 1:  {  leerEstudiante(estudiante);  insertarEstudiante(lista, estudiante, exito);  if (!exito) {  cout << "Lista llena: imposible insertar" << endl;  }  }  break;  Fundamentos de la programación: Tipos de datos estructurados Página 577  Luis Hernández Yáñez/Pablo Moreno Ger  case 2:  {  cout << "Posición: ";  cin >> pos;  eliminarEstudiante(lista, pos ‐ 1, exito);  if (!exito) {  cout << "Elemento inexistente!" << endl;  }  }  break;  case 3:  {  calificar(lista);  }  break;  case 4:  {  listado(lista, mediaClase(lista), mayorNota(lista));  }  }  } while (op != 0);  guardar(lista);  }  return 0;  }  Fundamentos de la programación: Tipos de datos estructurados Página 578  Luis Hernández Yáñez/Pablo Moreno Ger  *Licencia CC (Creative Commons)*  Este tipo de licencias ofrecen algunos derechos a terceras personas  bajo ciertas condiciones.  Este documento tiene establecidas las siguientes:  Pulsa en la imagen de arriba a la derecha para saber más.  Fundamentos de la programación: Tipos de datos estructurados Página 579  Reconocimiento (*Attribution*):  En cualquier explotación de la obra autorizada por la licencia  hará falta reconocer la autoría.  No comercial (*Non commercial*):  La explotación de la obra queda limitada a usos no comerciales.  Compartir igual (*Share alike*):  La explotación autorizada incluye la creación de obras derivadas  siempre que mantengan la misma licencia al ser divulgadas.  **5A**  Grado en Ingeniería Informática  Grado en Ingeniería del Software  Grado en Ingeniería de Computadores  Luis Hernández Yáñez/Pablo Moreno Ger  Facultad de Informática  Universidad Complutense  Fundamentos de la programación  Luis Hernández Yáñez/Pablo Moreno Ger  Fundamentos de la programación: Cadenas al estilo de C (Anexo)  Cadenas al estilo de C 582  E/S con cadenas al estilo de C 583  La biblioteca cstring 584  Ejemplo 585  Luis Hernández Yáñez/Pablo Moreno Ger  *Arrays de caracteres terminados en nulo*  const Max = 15;  typedef char tCadena[Max];  tCadena cadena = "Adiós"; // Inicialización al declarar  Siempre hay al final un carácter nulo (código ASCII 0 – '\0')  Indica que en esa posición termina la cadena (exclusive)  En el array caben MAX‐1 caracteres significativos  Longitud máxima de la variable cadena: 14  No se pueden asignar cadenas literales: cadena = "Hola";  Ni copiar cadenas directamente: cad2 = cad1;  Ni comparar con op. relacionales: if (cad1 < cad2) ...  Fundamentos de la programación: Cadenas al estilo de C (Anexo) Página 582  cadena A d i ó s \0  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14  Luis Hernández Yáñez/Pablo Moreno Ger  tCadena cadena;  cin >> cadena; // Se añade un nulo al final  Extractor: la lectura termina en el primer espacio en blanco  *¡No se comprueba si se leen más caracteres de los que caben!*  setw(): máximo de caracteres a colocar (incluyendo el nulo)  cin >> setw(15) >> cadena;  cin.getline(*cadena\_estilo\_C*, *máx*):  Para leer también los espacios en blanco y no más de *máx*‐1  cin.getline(cadena, 15); // Hasta 14 caracteres  cout << cadena << endl; // El nulo no se muestra  Fundamentos de la programación: Cadenas al estilo de C (Anexo) Página 583  cin.getline(*cad*, *máx*) Cadenas al estilo de C  getline(cin, *cad*) Cadenas de tipo string  Luis Hernández Yáñez/Pablo Moreno Ger   strlen(*cadena*): longitud actual de la *cadena*  cout << "Longitud: " << strlen(cadena);   strcpy(*destino*, *origen*): copia *origen* en *destino*  strcpy(cad2, cad1); strcpy(cad, "Me gusta C++");   strcat(*destino*, *origen*): añade *origen* al final de *destino*  tCadena cad1 = "Hola", cad2 = "Adiós";  strcat(cad1, cad2); // cad1 contiene "HolaAdiós"   strcmp(*cad1*, *cad2*): compara lexicográficamente las cadenas  0 si son iguales, 1 si *cad1* > *cad2* ó ‐1 si *cad1* < *cad2*  tCadena cad1 = "Hola", cad2 = "Adiós";  strcmp(cad1, cad2) // Devuelve 1 ("Hola" > "Adiós")  . . .  http://www.cplusplus.com/reference/clibrary/cstring/  Fundamentos de la programación: Cadenas al estilo de C (Anexo) Página 584  Luis Hernández Yáñez/Pablo Moreno Ger  #include <iostream>  using namespace std;  #include <cstring>  int main() {  const int MAX = 20;  typedef char tCad[MAX];  tCad cadena = "Me gusta C++";  cout << cadena << endl;  cout << "Cadena: ";  cin >> cadena; // Lee hasta el primer espacio en blanco  cout << cadena << endl;  cin.sync(); // Sincronizar la entrada  cout << "Cadena: ";  cin.getline(cadena, MAX);  cout << cadena << endl;  cout << "Longitud: " << strlen(cadena) << endl;  strcpy(cadena, "Hola");  ...  Fundamentos de la programación: Cadenas al estilo de C (Anexo) Página 585  cadenas.cpp  Luis Hernández Yáñez/Pablo Moreno Ger  tCad cadena2 = " amigo";  strcat(cadena, cadena2);  cout << cadena << endl;  if (strcmp(cadena, cadena2) == 0) {  cout << "Iguales";  }  else if (strcmp(cadena, cadena2) > 0) {  cout << cadena << " es mayor que " << cadena2;  }  else {  cout << cadena << " es menor que " << cadena2;  }  cout << endl;  return 0;  }  Fundamentos de la programación: Cadenas al estilo de C (Anexo) Página 586  Luis Hernández Yáñez/Pablo Moreno Ger  *Licencia CC (Creative Commons)*  Este tipo de licencias ofrecen algunos derechos a terceras personas  bajo ciertas condiciones.  Este documento tiene establecidas las siguientes:  Pulsa en la imagen de arriba a la derecha para saber más.  Fundamentos de la programación: Cadenas al estilo de C (Anexo) Página 587  Reconocimiento (*Attribution*):  En cualquier explotación de la obra autorizada por la licencia  hará falta reconocer la autoría.  No comercial (*Non commercial*):  La explotación de la obra queda limitada a usos no comerciales.  Compartir igual (*Share alike*):  La explotación autorizada incluye la creación de obras derivadas  siempre que mantengan la misma licencia al ser divulgadas.  **6**  Grado en Ingeniería Informática  Grado en Ingeniería del Software  Grado en Ingeniería de Computadores  Luis Hernández Yáñez / Pablo Moreno Ger  Facultad de Informática  Universidad Complutense  Fundamentos de la programación  Luis Hernández Yáñez  Fundamentos de la programación: Recorrido y búsqueda en arrays  Recorrido de arrays 590  Arrays completos 593  Arrays no completos con centinela 594  Arrays no completos con contador 595  Ejemplos 597  Generación de números aleatorios 601  Búsquedas en arrays 604  Arrays completos 606  Arrays no completos con centinela 607  Arrays no completos con contador 608  Ejemplo 610  Recorridos y búsquedas en cadenas 614  Más ejemplos de manejo de arrays 617  Arrays multidimensionales 630  Inicialización de arrays multidimensionales 638  Recorrido de un array bidimensional 641  Recorrido de un array N‐dimensional 644  Búsqueda en un array multidimensional 647  Luis Hernández Yáñez  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 590  Luis Hernández Yáñez  true  Finalización  *Esquema de recorrido*  Inicialización  Mientras no al final de la secuencia:  Obtener el siguiente elemento  Procesar el elemento  Finalización  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 591  Inicialización  false  Procesar elemento  Obtener elemento  ¿Al final?  Luis Hernández Yáñez  *Recorrido de secuencias en arrays*   Todas las posiciones ocupadas:  Tamaño del array = longitud de la secuencia  N elementos en un array de N posiciones:  Recorrer el array desde la primera posición hasta la última   Posiciones libres al final del array:  Tamaño del array > longitud de la secuencia   Con centinela:  Recorrer el array hasta encontrar el valor centinela   Con *contador* de elementos:  Recorrer el array hasta el índice *contador* – 1  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 592  Luis Hernández Yáñez  *Recorrido de arrays completos*  Todas las posiciones del array ocupadas  const int N = 10;  typedef double tVentas[N];  tVentas ventas;  ...  double elemento;  for (int i = 0; i < N; i++) {  elemento = ventas[i];  // Procesar el elemento ...  }  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 593  ventas 125.40 76.95 328.80 254.62 435.00 164.29 316.05 219.99 93.45 756.62  0 1 2 3 4 5 6 7 8 9  Luis Hernández Yáñez  *Recorrido de arrays no completos – con centinela*  No todas las posiciones del array están ocupadas  const int N = 10;  typedef double tArray[N];  tArray datos; // Datos positivos: centinela = ‐1  ...  int i = 0;  double elemento = datos[i];  while (elemento != ‐1) {  // Procesar el elemento ...  i++;  elemento = datos[i];  }  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 594  int i = 0;  double elemento;  do {  elemento = datos[i];  if (elemento != ‐1) {  // Procesar el elemento...  i++;  }  } while (elemento != ‐1);  datos 125.40 76.95 328.80 254.62 435.00 164.29 316.05 ‐1.0  0 1 2 3 4 5 6 7 8 9  Luis Hernández Yáñez  contador 7  *Recorrido de arrays no completos – con contador*  Array y contador íntimamente relacionados: estructura  const int N = 10;  typedef double tArray[N];  typedef struct {  tArray elementos;  int contador;  } tLista;  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 595  elementos  125.40 76.95 328.80 254.62 435.00 164.29 316.05  0 1 2 3 4 5 6 7 8 9  Nº de elementos (primer índice sin elemento)  Listas de elementos de longitud variable  Luis Hernández Yáñez  *Recorrido de arrays no completos – con contador*  const int N = 10;  typedef double tArray[N];  typedef struct {  tArray elementos;  int contador;  } tLista;  tLista lista;  ...  double elemento;  for (int i = 0; i < lista.contador; i++) {  elemento = lista.elementos[i];  // Procesar el elemento...  }  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 596  Luis Hernández Yáñez  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 597  Luis Hernández Yáñez  *Array con los N primeros números de Fibonacci*  const int N = 50;  typedef long long int tFibonacci[N]; // 50 números  tFibonacci fib;  fib[0] = 1;  fib[1] = 1;  for (int i = 2; i < N; i++) {  fib[i] = fib[i ‐ 1] + fib[i ‐ 2];  }  for (int i = 0; i < N; i++) {  cout << fib[i] << " ";  }  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 598  fibonacci.cpp  Luis Hernández Yáñez  *Cuenta de valores con k dígitos*  Recorrer una lista de N enteros contabilizando cuántos son  de 1 dígito, cuántos de 2 dígitos, etcétera (hasta 5 dígitos)  2 arrays: array con los números y array de contadores  const int NUM = 100;  typedef int tNum[NUM]; // Exactamente 100 números  tNum numeros;  const int DIG = 5;  typedef int tDig[DIG]; // i ‐‐> números de i+1 dígitos  tDig numDig = { 0 };  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 599  numeros 123 2 46237 2345 236 11234 33 999 ... 61  0 1 2 3 4 5 6 7 99  numDig 0 0 0 0 0 0  0 1 2 3 4 5  Luis Hernández Yáñez  *Cuenta de valores con k dígitos*  Función que devuelve el número de dígitos de un entero:  int digitos(int dato) {  int n\_digitos = 1; // Al menos tiene un dígito  // Recorremos la secuencia de dígitos...  while (dato >= 10) {  dato = dato / 10;  n\_digitos++;  }  return n\_digitos;  }  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 600  Luis Hernández Yáñez  *Generación de números pseudoaleatorios*  Probemos con una secuencia de enteros generada aleatoriamente  Función rand() (cstdlib): entero aleatorio entre 0 y 32766  srand() (cstdlib): inicia la secuencia de números aleatorios  Acepta un entero que usa como semilla para iniciar la secuencia  ¿Qué valor usar? Uno distinto en cada ejecución   El instante de tiempo actual (diferente cada vez)  Función time() (ctime): segundos transcurridos desde 1970  Requiere un argumento, que en nuestro caso será NULL  srand(time(NULL)); // Inicia la secuencia  ...  numeros[0] = rand(); // Entre 0 y 32766  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 601  Luis Hernández Yáñez  *Cuenta de valores con k dígitos*  #include <iostream>  using namespace std;  #include <cstdlib> // srand() y rand()  #include <ctime> // time()  int digitos(int dato);  int main() {  const int NUM = 100;  typedef int tNum[NUM]; // Exactamente 100 números  const int DIG = 5;  typedef int tDig[DIG];  tNum numeros;  tDig numDig = { 0 }; // Inicializa todo el array a 0  srand(time(NULL)); // Inicia la secuencia aleatoria  ...  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 602  digitos.cpp  Luis Hernández Yáñez  for (int i = 0; i < NUM; i++) { // Creamos la secuencia  numeros[i] = rand(); // Entre 0 y 32766  }  for (int i = 0; i < NUM; i++) {  // Recorremos la secuencia de enteros  numDig[digitos(numeros[i]) ‐ 1]++;  }  for (int i = 0; i < DIG; i++) {  // Recorremos la secuencia de contadores  cout << "De " << i + 1 << " díg. = " << numDig[i]  << endl;  }  return 0;  }  int digitos(int dato) {  ...  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 603  Luis Hernández Yáñez  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 604  Luis Hernández Yáñez  *Esquema de búsqueda*  Inicialización  Mientras no se encuentre el elemento  y no se esté al final de la secuencia:  Obtener el siguiente elemento  Comprobar si el elemento  satisface la condición  Finalización  (tratar el elemento encontrado  o indicar que no se ha encontrado)  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 605  false  true  Inicialización / encontrado = false;  ¿Encontrado?  ¿Al final o  encontrado?  Obtener elemento  Finalización  Luis Hernández Yáñez  Todas las posiciones ocupadas  int buscado;  bool encontrado = false;  cout << "Valor a buscar: ";  cin >> buscado;  int pos = 0;  while ((pos < N) && !encontrado) {  // Mientras no se llegue al final y no encontrado  if (lista[pos] == buscado) {  encontrado = true;  }  else {  pos++;  }  }  if (encontrado) // ...  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 606  const int N = 100;  typedef int tArray[N];  tArray lista;  Luis Hernández Yáñez  *Con centinela*  int buscado;  cout << "Valor a buscar: ";  cin >> buscado;  int pos = 0;  bool encontrado = false;  while ((array[pos] != centinela) && !encontrado) {  if (array[pos] == buscado) {  encontrado = true;  }  else {  pos++;  }  }  if (encontrado) // ...  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 607  const int N = 10;  typedef int tArray[N];  tArray array;  const int centinela = ‐1;  Luis Hernández Yáñez  *Con contador*  int buscado;  cout << "Valor a buscar: ";  cin >> buscado;  int pos = 0;  bool encontrado = false;  while ((pos < miLista.contador)  && !encontrado) {  // Mientras no al final y no encontrado  if (miLista.elementos[pos] == buscado) {  encontrado = true;  }  else {  pos++;  }  }  if (encontrado) // ...  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 608  const int N = 10;  typedef double tArray[N];  typedef struct {  tArray elementos;  int contador;  } tLista;  tLista miLista;  Luis Hernández Yáñez  *Acceso directo a cualquier posición*  Acceso directo: *array*[*posición*]  Si se puede calcular la posición del elemento, su acceso es directo  typedef double tVentaMes[DIAS][SUCURSALES];  typedef struct {  tVentaMes ventas;  int dias;  } tMes;  typedef tMes tVentaAnual[MESES];  tVentaAnual anual;  *Ventas del cuarto día del tercer mes en la primera sucursal:*  anual[2].ventas[3][0]  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 609  Luis Hernández Yáñez  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 610  Luis Hernández Yáñez  #include <iostream>  using namespace std;  #include <fstream>  const int N = 100;  typedef double tArray[N];  typedef struct {  tArray elementos;  int contador;  } tLista;  void cargar(tLista &lista, bool &ok);  int main() {  tLista lista;  bool ok;  cargar(lista, ok);  if (!ok) {  cout << "Error: no hay archivo o demasiados datos"  << endl;  }  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 611  umbral.cpp  Luis Hernández Yáñez  else {  double umbral;  cout << "Valor umbral: "; cin >> umbral;  bool encontrado = false;  int pos = 0;  while ((pos < lista.contador) && !encontrado) {  if (lista.elementos[pos] > umbral) {  encontrado = true;  }  else {  pos++;  }  }  if (encontrado) {  cout << "Valor en pos. " << pos + 1 << " ("  << lista.elementos[pos] << ")" << endl;  }  else {  cout << "¡No encontrado!" << endl;  }  }  return 0;  }  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 612  Luis Hernández Yáñez  void cargar(tLista &lista, bool &ok) {  ifstream archivo;  double dato;  bool abierto = true, overflow = false;  lista.contador = 0;  archivo.open("datos.txt");  if (!archivo.is\_open()) {  abierto = false;  }  else {  archivo >> dato;  while ((dato >= 0) && !overflow) {  if (lista.contador == N) {  overflow = true; // ¡Demasiados!  }  else {  lista.elementos[lista.contador] = dato;  lista.contador++;  archivo >> dato;  }  }  archivo.close();  }  ok = abierto && !overflow;  }  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 613  Luis Hernández Yáñez  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 614  Luis Hernández Yáñez  *Recorridos y búsquedas en cadenas de caracteres*  Longitud de la cadena: size() o length()  Caso similar a los arrays con contador de elementos  Ejemplo: Recorrido de una cadena generando otra invertida  string cadena, inversa = "";  int pos;  char car;  // ... (lectura de cadena)  pos = 0;  while (pos < cadena.size()) {  // Mientras no se llegue al final de la cadena  car = cadena.at(pos);  inversa = car + inversa; // Inserta car al principio  pos++;  } // ...  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 615  inversa.cpp  Luis Hernández Yáñez  *Búsqueda de un carácter en una cadena*  string cadena;  char buscado;  int pos;  bool encontrado;  // ... (lectura de cadena)  cout << "Introduce el carácter a buscar: ";  cin >> buscado;  pos = 0;  encontrado = false;  while ((pos < cadena.size()) && !encontrado) {  if (cadena.at(pos) == buscado) {  encontrado = true;  }  else {  pos++;  }  }  if (encontrado) // ...  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 616  busca.cpp  Luis Hernández Yáñez  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 617  Luis Hernández Yáñez  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 618  Tipo tVector para representar secuencias de N enteros:  const int N = 10;  typedef int tVector[N];  Subprogramas:   Dado un vector, mueve sus componentes un lugar a la derecha;  el último componente se moverá al 1er lugar   Dado un vector, calcula y devuelve la suma de los elementos que se  encuentran en las posiciones pares del vector   Dado un vector, encuentra y devuelve la componente mayor   Dados dos vectores, devuelve un valor que indique si son iguales   Dado un vector, determina si alguno de los valores almacenados en  el vector es igual a la suma del resto de los valores del mismo;  devuelve el índice del primero encontrado o ‐1 si no se encuentra   Dado un vector, determina si alguno de los valores almacenados  en el vector está repetido  Luis Hernández Yáñez  void desplazar(tVector v) {  int aux = v[N ‐ 1];  for (int i = N ‐ 1; i > 0; i‐‐) {  v[i] = v[i ‐ 1];  }  v[0] = aux;  }  int sumaPares(const tVector v) {  int suma = 0;  for (int i = 0; i < N; i = i + 2) {  suma = suma + v[i];  }  return suma;  }  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 619  vectores.cpp  Luis Hernández Yáñez  int encuentraMayor(const tVector v) {  int max = v[0], posMayor = 0;  for (int i = 1; i < N; i++) {  if (v[i] > max) {  posMayor = i;  max = v[i];  }  }  return posMayor;  }  bool sonIguales(const tVector v1, const tVector v2) {  //Implementación como búsqueda del primer elemento distinto  bool encontrado = false;  int i = 0;  while ((i<N) && !encontrado) {  encontrado = (v1[i] != v2[i]);  i++;  }  return !encontrado;  }  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 620  Luis Hernández Yáñez  int compruebaSuma(const tVector v) {  // ¿Alguno igual a la suma del resto?  bool encontrado = false;  int i = 0;  int suma;  while ((i < N) && !encontrado) {  suma = 0;  for (int j = 0; j < N; j++) {  if (j != i) {  suma = suma + v[j];  }  }  encontrado = (suma == v[i]);  i++;  }  if (!encontrado) {  i = 0;  }  return i ‐ 1;  }  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 621  Luis Hernández Yáñez  bool hayRepetidos(const tVector v) {  bool encontrado = false;  int i = 0, j;  while ((i < N) && !encontrado) {  j = i + 1;  while ((j < N) && !encontrado) {  encontrado = (v[i] == v[j]);  j++;  }  i++;  }  return encontrado;  }  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 622  Luis Hernández Yáñez  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 623  Dado un vector de N caracteres v1, en el que no hay elementos  repetidos, y otro vector de M caracteres v2, donde N ≤ M, se  quiere comprobar si todos los elementos del vector v1 están  también en el vector v2  Por ejemplo, si:  v1= 'a' 'h' 'i' 'm'  v2= 'h' 'a' 'x' 'x' 'm' 'i'  El resultado sería cierto, ya que todos los elementos de v1 están  en v2  Luis Hernández Yáñez  #include <iostream>  using namespace std;  const int N = 3;  const int M = 10;  typedef char tVector1[N];  typedef char tVector2[M];  bool esta(char dato, const tVector2 v2);  bool vectorIncluido(const tVector1 v1, const tVector2 v2);  int main() {  tVector1 v1 = { 'a', 'b', 'c' };  tVector2 v2 = { 'a', 'r', 'e', 't', 'z', 's', 'a', 'h', 'b', 'x' };  bool ok = vectorIncluido(v1, v2);  if (ok) {  cout << "OK: v1 esta incluido en v2" << endl;  }  else {  cout << "NO: v1 no esta incluido en v2" << endl;  }  return 0;  }  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 624  incluidos.cpp  Luis Hernández Yáñez  bool esta(char dato, const tVector2 v2) {  int i = 0;  bool encontrado = (dato == v2[0]);  while (!encontrado && (i < M ‐ 1)) {  i++;  encontrado = (dato == v2[i]);  }  return encontrado;  }  bool vectorIncluido(const tVector1 v1, const tVector2 v2) {  int i = 0;  bool encontrado = esta(v1[0], v2);  while (encontrado && (i < N ‐ 1)) {  i++;  encontrado = esta(v1[i], v2);  }  return encontrado;  }  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 625  Luis Hernández Yáñez  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 626  Un programa que lea dos cadenas del teclado y determine si una es  un anagrama de la otra, es decir, si una cadena es una permutación  de los caracteres de la otra.  Por ejemplo, "acre" es un anagrama de "cera" y de "arce". Ten  en cuenta que puede haber letras repetidas ("carro", "llave").  Luis Hernández Yáñez  #include <iostream>  #include <string>  using namespace std;  int buscaCaracter(string cad, char c); // Índice o ‐1 si no está  int main() {  string cad1, cad2;  bool sonAnagramas = true;  int numCar, posEnCad2;  cout << "Introduce la primera cadena: ";  getline(cin, cad1);  cout << "Introduce la segunda cadena: ";  getline(cin, cad2);  if (cad1.length() != cad2.length()) { // No son anagramas  sonAnagramas = false;  }  else {  numCar = 0; // Contador de caracteres de la primera cadena  while (sonAnagramas && (numCar < cad1.length())) {  posEnCad2 = buscaCaracter(cad2, cad1.at(numCar));  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 627  anagramas.cpp  Luis Hernández Yáñez  if (posEnCad2 == ‐1) { //No se ha encontrado el caracter  sonAnagramas = false;  }  else {  cad2.erase(posEnCad2, 1);  }  numCar++;  }  }  if (sonAnagramas) {  cout << "Las palabras introducidas son anagramas" << endl;  }  else {  cout << "Las palabras introducidas NO son anagramas" << endl;  }  return 0;  }  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 628  Luis Hernández Yáñez  int buscaCaracter(string cad, char c) {  int pos = 0, lon = cad.length();  bool encontrado = false;  while ((pos < lon) && !encontrado) {  if (cad.at(pos) == c) {  encontrado = true;  }  else {  pos++;  }  }  if (!encontrado) {  pos = ‐1;  }  return pos;  }  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 629  Luis Hernández Yáñez  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 630  Luis Hernández Yáñez  *Arrays de varias dimensiones*  Varios tamaños en la declaración: cada uno con sus corchetes  typedef *tipo\_base nombre*[*tamaño1*][*tamaño2*]...[*tamañoN*];  Varias dimensiones, tantas como tamaños se indiquen  typedef double tMatriz[50][100];  tMatriz matriz;  Tabla bidimensional de 50 filas por 100 columnas:  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 631  0 1 2 3 ... 98 99  0 ...  1 ...  2 ...  ... ... ... ... ... ... ... ...  48 ...  49 ...  Luis Hernández Yáñez  0 1 2 3 ... 98 99  0 ...  1 ...  2 ...  ... ... ... ... ... ... ... ...  48 ...  49 ...  *Arrays de varias dimensiones*  typedef double tMatriz[50][100];  tMatriz matriz;  Cada elemento se localiza con dos índices, uno por dimensión  cout << matriz[2][98];  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 632  Luis Hernández Yáñez  *Arrays de varias dimensiones*  Podemos definir tantas dimensiones como necesitemos  typedef double tMatriz[5][10][20][10];  tMatriz matriz;  Necesitaremos tantos índices como dimensiones:  cout << matriz[2][9][15][6];  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 633  Luis Hernández Yáñez  *Ejemplo de array bidimensional*  Temperaturas mínimas y máximas  Matriz bidimensional de días y mínima/máxima:  const int MaxDias = 31;  const int MED = 2; // Nº de medidas  typedef double tTemp[MaxDias][MED]; // Día x mín./máx.  tTemp temp;  Ahora:   temp[i][0] es la temperatura mínima del día i+1   temp[i][1] es la temperatura máxima del día i+1  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 634  Luis Hernández Yáñez  int main() {  const int MaxDias = 31;  const int MED = 2; // Nº de medidas  typedef double tTemp[MaxDias][MED]; // Día x mín./máx.  tTemp temp;  double tMaxMedia = 0, tMinMedia = 0,  tMaxAbs = ‐100, tMinAbs = 100;  int dia = 0;  double max, min;  ifstream archivo;  archivo.open("temp.txt");  if (!archivo.is\_open()) {  cout << "No se ha podido abrir el archivo!" << endl;  }  else {  archivo >> min >> max;  // El archivo termina con ‐99 ‐99  ...  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 635  temp.cpp  Luis Hernández Yáñez  while (!((min == ‐99) && (max == ‐99))  && (dia < MaxDias)) {  temp[dia][0] = min;  temp[dia][1] = max;  dia++;  archivo >> min >> max;  }  archivo.close();  for (int i = 0; i < dia; i++) {  tMinMedia = tMinMedia + temp[i][0];  if (temp[i][0] < tMinAbs) {  tMinAbs = temp[i][0];  }  tMaxMedia = tMaxMedia + temp[i][1];  if (temp[i][1] > tMaxAbs) {  tMaxAbs = temp[i][1];  }  }  ...  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 636  Luis Hernández Yáñez  tMinMedia = tMinMedia / dia;  tMaxMedia = tMaxMedia / dia;  cout << "Temperaturas mínimas.‐" << endl;  cout << " Media = " << fixed << setprecision(1)  << tMinMedia << " C Mínima absoluta = "  << setprecision(1) << tMinAbs << " C" << endl;  cout << "Temperaturas máximas.‐" << endl;  cout << " Media = " << fixed << setprecision(1)  << tMaxMedia << " C Máxima absoluta = "  << setprecision(1) << tMaxAbs << " C" << endl;  }  return 0;  }  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 637  Luis Hernández Yáñez  Podemos dar valores a los elementos de un array al declararlo  Arrays bidimensionales:  typedef int tArray[5][2];  tArray cuads = {1,1, 2,4, 3,9, 4,16, 5,25};  Se asignan en el orden en el que los elementos están en memoria  La memoria es de una dimensión: secuencia de celdas  En memoria varían más rápidamente los índices de la derecha:  cuads[0][0] cuads[0][1] cuads[1][0] cuads[1][1] cuads[2][0]...  Para cada valor del primer índice: todos los valores del segundo  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 638  Luis Hernández Yáñez  *Inicialización de un array bidimensional*  typedef int tArray[5][2];  tArray cuads = {1,1, 2,4, 3,9, 4,16, 5,25};  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 639  Memoria  cuads[0][0] **1**  cuads[0][1] **1**  cuads[1][0] **2**  cuads[1][1] **4**  cuads[2][0] **3**  cuads[2][1] **9**  cuads[3][0] **4**  cuads[3][1] **16**  cuads[4][0] **5**  cuads[4][1] **25**  Si hay menos valores que elementos,  el resto se inicializan a cero  Inicialización a cero de todo el array:  int cuads[5][2] = { 0 };  0 1  0 1 1  1 2 4  2 3 9  3 4 16  4 5 25  Luis Hernández Yáñez  typedef double tMatriz[3][4][2][3];  tMatriz matriz =  {1, 2, 3, 4, 5, 6,  7, 8, 9, 10, 11, 12};  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 640  Memoria  matriz[0][0][0][0] **1**  matriz[0][0][0][1] **2**  matriz[0][0][0][2] **3**  matriz[0][0][1][0] **4**  matriz[0][0][1][1] **5**  matriz[0][0][1][2] **6**  matriz[0][1][0][0] **7**  matriz[0][1][0][1] **8**  matriz[0][1][0][2] **9**  matriz[0][1][1][0] **10**  matriz[0][1][1][1] **11**  matriz[0][1][1][2] **12**  matriz[0][2][0][0] **0**  ... 0  Luis Hernández Yáñez  const int FILAS = 10;  const int COLUMNAS = 5;  typedef double tMatriz[FILAS][COLUMNAS];  tMatriz matriz;  Para cada *fila* (de 0 a FILAS – 1):  Para cada *columna* (de 0 a COLUMNAS – 1):  Procesar el elemento en [*fila*][*columna*]  for (int fila = 0; fila < FILAS; fila++) {  for (int columna = 0; columna < COLUMNAS; columna++) {  // Procesar matriz[fila][columna]  }  }  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 641  Luis Hernández Yáñez  *Ventas de todos los meses de un año*  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 642  const int Meses = 12;  const int MaxDias = 31;  typedef double tVentas[Meses][MaxDias];  tVentas ventas; // Ventas de todo el año  typedef short int tDiasMes[Meses];  tDiasMes diasMes;  inicializa(diasMes); // Nº de días de cada mes  // Pedimos las ventas de cada día del año...  for (int mes = 0; mes < Meses; mes++) {  for (int dia = 0; dia < diasMes[mes]; dia++) {  cout << "Ventas del día " << dia + 1  << " del mes " << mes + 1 << ": ";  cin >> ventas[mes][dia];  }  }  Luis Hernández Yáñez  *Ventas de todos los meses de un año*  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 643  0 1 2 3 4 ... 28 29 30  0 201 125 234 112 156 ... 234 543 667  1 323 231 675 325 111 ...  2 523 417 327 333 324 ... 444 367 437  3 145 845 654 212 562 ... 354 548  4 327 652 555 222 777 ... 428 999 666  5 854 438 824 547 175 ... 321 356  6 654 543 353 777 437 ... 765 678 555  7 327 541 164 563 327 ... 538 159 235  8 333 327 432 249 777 ... 528 529  9 524 583 333 100 334 ... 743 468 531  10 217 427 585 218 843 ... 777 555  11 222 666 512 400 259 ... 438 637 879  Días  Meses  Celdas no  utilizadas  Luis Hernández Yáñez  const int DIM1 = 10;  const int DIM2 = 5;  const int DIM3 = 25;  const int DIM4 = 50;  typedef double tMatriz[DIM1][DIM2][DIM3][DIM4];  tMatriz matriz;  Bucles anidados, desde la primera dimensión hasta la última:  for (int n1 = 0; n1 < DIM1; n1++) {  for (int n2 = 0; n2 < DIM2; n2++) {  for (int n3 = 0; n3 < DIM3; n3++) {  for (int n4 = 0; n4 < DIM4; n4++) {  // Procesar matriz[n1][n2][n3][n4]  }  }  }  }  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 644  Luis Hernández Yáñez  *Ventas diarias de cuatro sucursales*  Cada mes del año: ingresos de cada sucursal cada día del mes  Meses con distinto nº de días  junto con la matriz de ventas  mensual guardamos el nº de días del mes concreto  estructura  const int DIAS = 31;  const int SUCURSALES = 4;  typedef double tVentaMes[DIAS][SUCURSALES];  typedef struct {  tVentaMes ventas;  int dias;  } tMes;  const int MESES = 12;  typedef tMes tVentaAnual[MESES];  tVentaAnual anual;  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 645  anual  tVentaAnual  anual[i]  tMes  anual[i].dias  int  anual[i].ventas  tVentaMes  anual[i].ventas[j][k]  double  Luis Hernández Yáñez  Cálculo de las ventas  de todo el año:  Para cada mes...  Para cada día del mes...  Para cada sucursal...  Acumular las ventas  double total = 0;  for (int mes = 0; mes < MESES; mes++) {  for (int dia = 0; dia < anual[mes].dias; dia++) {  for (int suc = 0; suc < SUCURSALES; suc++) {  total = total + anual[mes].ventas[dia][suc];  }  }  }  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 646  const int DIAS = 31;  const int SUCURSALES = 4;  typedef double  tVentaMes[DIAS][SUCURSALES];  typedef struct {  tVentaMes ventas;  int dias;  } tMes;  const int MESES = 12;  typedef tMes tVentaAnual[MESES];  tVentaAnual anual;  Luis Hernández Yáñez  bool encontrado = false;  int mes = 0, dia, suc;  while ((mes < MESES) && !encontrado) {  dia = 0;  while ((dia < anual[mes].dias) && !encontrado) {  suc = 0;  while ((suc < SUCURSALES) && !encontrado) {  if (anual[mes].ventas[dia][suc] > umbral) {  encontrado = true;  }  else {  suc++;  }  }  if (!encontrado) {  dia++;  }  }  if (!encontrado) {  mes++;  }  }  if (encontrado) { ...  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 647  Primer valor > umbral  Luis Hernández Yáñez  *Licencia CC (Creative Commons)*  Este tipo de licencias ofrecen algunos derechos a terceras personas  bajo ciertas condiciones.  Este documento tiene establecidas las siguientes:  Pulsa en la imagen de arriba a la derecha para saber más.  Fundamentos de la programación: Recorrido y búsqueda en arrays Página 648  Reconocimiento (*Attribution*):  En cualquier explotación de la obra autorizada por la licencia  hará falta reconocer la autoría.  No comercial (*Non commercial*):  La explotación de la obra queda limitada a usos no comerciales.  Compartir igual (*Share alike*):  La explotación autorizada incluye la creación de obras derivadas  siempre que mantengan la misma licencia al ser divulgadas.  **7**  Grado en Ingeniería Informática  Grado en Ingeniería del Software  Grado en Ingeniería de Computadores  Luis Hernández Yáñez  Facultad de Informática  Universidad Complutense  Fundamentos de la programación  Luis Hernández Yáñez  Fundamentos de la programación: Algoritmos de ordenación |

## Programas y algoritmos

Un algoritmo es una secuencia no ambigua, finita y ordenada de instrucciones que han de seguirse para resolver un problema. Un programa normalmente implementa (traduce a un lenguaje de programación concreto) un algoritmo. Nótese que es la secuencia de instrucciones en sí la que debe ser finita, no el número de pasos realizados como la ejecución de ellas.

Los programas suelen subdividirse en partes menores (módulos), de modo que la complejidad algorítmica de cada una de las partes sea menor que la del programa completo, lo cual ayuda al desarrollo del programa.

Según Niklaus Wirth un programa está formado por algoritmos y estructura de datos.

Se han propuesto diversas técnicas de programación, cuyo objetivo es mejorar tanto el proceso de creación de software como su mantenimiento. Entre ellas se pueden mencionar las programaciones lineal, estructurada, modular y orientada a objetos.

## Compilación

El programa escrito en un lenguaje de programación (comprensible por el ser humano, aunque se suelen corresponder con lenguajes formales descritos por gramáticas independientes del contexto) no es inmediatamente ejecutado en una computadora. La opción más común es compilar el programa, aunque también puede ser ejecutado mediante un intérprete informático

El código fuente del programa se debe someter a un proceso de transformación para convertirse en lenguaje máquina, interpretable por el procesador. A este proceso se le llama compilación.

Normalmente la creación de un programa ejecutable (un típico.exe para Microsoft Windows) conlleva dos pasos. El primer paso se llama compilación (propiamente dicho) y traduce el código fuente escrito en un lenguaje de programación almacenado en un archivo a código en bajo nivel, (normalmente en código objeto no directamente al lenguaje máquina). El segundo paso se llama enlazado (del inglés *link* o *linker*) se junta el código de bajo nivel generado de todos los ficheros que se han mandado compilar y se añade el código de las funciones que hay en las bibliotecas del compilador para que el ejecutable pueda comunicarse con el sistemas operativo y traduce el código objeto a código máquina.

Estos dos pasos se pueden mandar hacer por separado, almacenando el resultado de la fase de compilación en archivos objetos (un típico.obj para Microsoft Windows,.o para Unix), para enlazarlos posteriormente, o crear directamente el ejecutable con lo que la fase de compilación se almacena sólo temporalmente. Un programa podría tener partes escritas en varios lenguajes (generalmente C, C++ y Asm), que se podrían compilar de forma independiente y enlazar juntas para formar un único ejecutable.

## Programación e ingeniería del software

Existe una tendencia a identificar el proceso de creación de un programa informático con la programación, que es cierta cuando se trata de programas pequeños para uso personal, y que dista de la realidad cuando se trata de grandes proyectos.

El proceso de creación de software desde el punto de vista de la Ingeniería tiene los siguientes pasos:

1. Reconocer la necesidad de un programa para solucionar un problema ó identificar la posibilidad de automatización de una tarea.
2. Recoger los requisitos del programa. Debe quedar claro qué es lo que debe hacer el programa y para qué se necesita.
3. Realizar el análisis de los requisitos del programa. Debe quedar claro cómo debe realizar el programa las cosas que debe hacer. Las pruebas que comprueben la validez del programa se pueden especificar en esta fase.
4. Diseñar la arquitectura del programa. Se debe descomponer el programa en partes de complejidad abordable.
5. Implementar el programa. Consiste en realizar un diseño detallado, especificando completamente todo el funcionamiento del programa, tras lo cual la codificación debería resultar inmediata.
6. Implantar (instalar) el programa. Consiste en poner el programa en funcionamiento junto con los componentes que pueda necesitar (bases de datos, redes de comunicaciones, etc.)

La Ingeniería del Software se centra en los pasos de planificación y diseño del programa, mientras que antiguamente (programación artesanal) la realización de un programa consistía únicamente en escribir el código.

## Objetivos de la programación

La programación de ordenadores debe perseguir tres objetivos fundamentales:

* Corrección: un programa es correcto si hace lo que debe hacer. Para determinar si un programa hace lo que debe es muy importante especificar claramente qué debe hacer el programa antes de desarrollarlo y una vez acabado compararlo con lo que realmente hace.
* Claridad: es muy importante que el programa sea lo más claro y legible posible para mejorar el mantenimiento del software. Cuando se acaba de escribir el código del programa, se deben buscar errores y corregirlos. Más concretamente, cuando el programa está concluido, es nec