LICEO COMPU-MARKET

Catedrático: Erick Gonzalo

Cátedra: Practica Supervisada

Programación

Nombre: Nora Elizabeth Sánchez Oliveros

Grado: 5to Bachillerato en computación

Con orientación científica

Fecha: 21/04/2017

**INDICE**

Contenido

[qué debían calcular. 5](#_Toc480306842)

[Arquitectura de hardware 7](#_Toc480306843)

[Dispositivos de salida 7](#_Toc480306844)

[Dispositivos de almacenamiento. 7](#_Toc480306845)

[Dispositivos de comunicación 8](#_Toc480306846)

[Dispositivo de cómputo: 8](#_Toc480306847)

[Arquitectura de software 9](#_Toc480306848)

[Software: 9](#_Toc480306849)

[Lenguaje de programación: 9](#_Toc480306850)

[Programa: 9](#_Toc480306851)

[Sistema Operativo 9](#_Toc480306852)

[OBJETIVOS 11](#_Toc480306853)

[INTRODUCCIÓN 11](#_Toc480306854)

[ALGORITMOS 11](#_Toc480306855)

[Representación de Algoritmos 14](#_Toc480306856)

[A Diagramas de flujo 15](#_Toc480306857)

[Pseudocódigo 15](#_Toc480306858)

[Teoría de la Programación Estructurada 16](#_Toc480306859)

[Teorema de la estructura. 16](#_Toc480306860)

[OBJETIVO 18](#_Toc480306861)

[INTRODUCCIÓN 19](#_Toc480306862)

[abstracciones o subprogramas llamados procedimientos y funciones. Lo que debe hacer el 23](#_Toc480306863)

[Operadores aritméticos 52](#_Toc480306864)

[LAS VARIABLES NO SE INICIALIZAN 53](#_Toc480306865)

**INTRODUCCION**

Se llama Programación a la implementación de un algoritmo en un determinado lenguaje de programación, para realizar un programa.

Algoritmo es una secuencia no ambigua, finita y ordenada de instrucciones que han de seguirse para resolver un problema.

Programa (Software en inglés) es una secuencia de instrucciones que una computadora puede interpretar y ejecutar.

El proceso de creación de software es materia de la ingeniería del software, una de las ramas propias de la Ingeniería Informática.

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.

HISTORIA DE LA COMPUTACIÓN

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.

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.  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”.  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  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”* 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  **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.  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.  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  Dispositivos de comunicación**:** Son aquellos que le permiten a un computador comunicarse  con otros. Entre estos se cuentan los modems y tarjetas de red.  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.  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  **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. 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. 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.  ***Lenguajes a diferente nivel***  ***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:  Lenguaje de máquina  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:  Un **ENSAMBLADOR** es un software, generalmente escrito en lenguaje de máquina, que es  capaz de traducir de lenguaje ensamblador a lenguaje de máquina.  11  **Lenguaje Ensamblador.**  ***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.  **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  Arquitectura de Software Completa.  **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.  NTRODUCCIÓN A LOS ALGORITMOS 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. 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. 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.  ***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.  ***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.  ***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 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: 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. 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***. Teoría de la Programación EstructuradaTeorema 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  ***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  METODOLOGÍA DE SOLUCIÓN DE PROBLEMAS CON EL COMPUTADOR 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. 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  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*.  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:  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.  *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  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:  Leer un dato.  Determinar si el elemento no está en el primer conjunto.  **.** Si no está, agregar el dato al primer conjunto. Si ya está, mostrar un mensaje de error.  **.** Preguntar si el usuario desea ingresar un nuevo elemento al primer conjunto.  Si el usuario desea ingresar un nuevo elemento volver a 1.1. Si no, continuar.  Leer segundo conjunto se divide en:  35  Leer un dato.  **.** Determinar si el elemento no está en el segundo conjunto.  **.** Si no está, agregar el dato al segundo conjunto. Si ya está, mostrar un mensaje de error.  **.** Preguntar si el usuario desea ingresar un nuevo elemento al segundo conjunto.  **.** Si el usuario desea ingresar un nuevo elemento volver a 2.1. Sino, continuar.  Leer entero.  Determinar si el entero está en el primer conjunto.  Determinar si el entero está al segundo conjunto.  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.  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( intA[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  Parque Jurásico  Luis Hernández Yáñez  *Esquema general*  Fundamentos de la programación: Computadoras y programación  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  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  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  1 Kilobyte (KB) = 1024 Bytes  Fundamentos de la programación: Computadoras y programación  *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  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  Código objeto  (lenguaje máquina)  Programa  ensamblador  Código fuente  (lenguaje ensamblador)  Fundamentos de la programación: Computadoras y programación   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  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  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  *Genealogía de lenguajes*  BASIC  1964  LaLa *prehistoria*  El ábaco  Siglo XIX  Máquina analítica de Charles Babbage  Fundamentos de la programación:  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  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  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  Luis Hernández Yáñez  Fundamentos de la programación: Computadoras y programación  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  Entrada  *La programación es sólo una etapa del proceso de desarrollo*  Modelo de desarrollo “en cascada”:  Fundamentos de la programación: Computadoras y programación  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  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  Hola Mundo!  Luis Hernández Yáñez  Instrucciones  Datos: literales, variables, tipos  Subprogramas (funciones)  Comentarios  Directivas  ...  Fundamentos de la programación: Computadoras y programación  #include <iostream>  using namespace std;  int main()  {  cout << "Hola Mundo!" << endl;  // Muestra Hola Mundo!  return 0;  }  Directiva  ComentarioFundamentos de la programación: Computadoras y programación  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  Luis Hernández Yáñez  *Especificación*   Lenguajes (BNF)   Diagramas  Fundamentos de la programación: Computadoras y programación  <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  ‐  137  Ejemplo: Números enteros (sin decimales)  Fundamentos de la programación: Computadoras y programación  <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>)        Fundamentos de la programación: Computadoras y programación  +23   1374   ? 1‐   +23  1374  1‐34  **+**  **0 .. 9**  **+**  **+**  **‐**  **‐**  **‐**  **0 .. 9**  **0 .. 9**  Luis Hernández Yáñez  Fundamentos de la programación: Computadoras y programación  *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  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  Luis Hernández Yáñez  Fundamentos de la programación: Computadoras y programación  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  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  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  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  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  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  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  Luis Hernández Yáñez  *Sintaxis del lenguaje de programación*  Fundamentos de la programación: Tipos e instrucciones  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  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  *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  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  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  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  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;  *La ejecución*  Se realiza lo que pide  cada instrucción:  Fundamentos de la programación: Tipos e instrucciones  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  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  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;  ?  *Depuración*  Editamos el código para arreglar el error lógico:  Fundamentos de la programación: Tipos e instrucciones  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  **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  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  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  Luis Hernández Yáñez  cout (iostream)  Visualización en la pantalla: operador << (*insertor*)  Fundamentos de la programación: Tipos e instrucciones 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  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  **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  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  *¡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  Luis Hernández Yáñez  *El programa principal*  La función main():  int main()  {  ...  return 0;  }  Fundamentos de la programación: Tipos e instrucciones  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  *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  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  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  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  return  ¿Cuál se lee mejor?  Fundamentos de la programación: Tipos e instrucciones  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones  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  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  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  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  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  ó  3 esp.  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones 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  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  *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  cálculos.cpp  Un texto Un número  Fundamentos de la programación: Tipos e instrucciones  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  Fundamentos de la programación: Tipos e instrucciones  *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  *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  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  Memoria  cantidad **?**  precio **?**  total **?**  **...**  Podemos declarar varias de un mismo tipo  separando los nombres con comas  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  *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  Instrucción: termina en ;  cantidad  12  Fundamentos de la programación: Tipos e instrucciones  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  *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  *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  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 **?**  *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  Memoria  **...**  cantidad **12**  precio **?**  total **?**  *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  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  Memoria  **...**  cantidad **12**  precio **39.95**  total **479.4**  *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  Memoria  **...**  cantidad **12**  precio **39.95**  total **479.4**  *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  Luis Hernández Yáñez  Fundamentos de la programación: Tipos e instrucciones  Luis Hernández Yáñez  cin (iostream)  Lectura de valores de variables: operador >> (*extractor*)  Fundamentos de la programación: Tipos e instrucciones  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  cin  *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?  *¿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!!!*  *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  *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;  *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  \_  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  *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  *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  *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?  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  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  *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  Z, \_  Fundamentos de la programación: Tipos e instrucciones I Página 133  *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*  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      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*  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  "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*  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?*  — 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)  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*  *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  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  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;  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  *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  *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  *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;  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;  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()  **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 peOOr información  **1**  **!**  **a**  **l**  **o**  **H**  **X**  Mediogg  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  nota2.cpp  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  *elemento*  Obtener siguiente  longitud.cpp  cont.cpp  Finalización  ¿Encontrado  o  i<Dias  )  \* / %O  < <= > >  == !=  &&  |  ?:  = += ‐= \*= /= %  ? : *Exp2*  primos2.cpp  Aumentan el nivel de abstracción del programa  Facilitan la prueba, la depuración y  conversiones.c  12.40 10.96 843 11.65 13.70 1.41 14.07 |