**El Software**

**Lenguajes de programación**

**Resolución de problemas con computadora**

**Entorno de programación**

**Tipos de Datos**

**Variables y Constantes**

**Sentencias**

**Operadores y Expresiones**

**Estructuras Básicas de Control**

**Procedimientos y Funciones**

**Visibilidad de variables**

**Bibliotecas**

**Arrays**

**El Estilo de Programación**

**El Software**

Las operaciones que debe realizar el hardware son especificadas con una lista de instrucciones, llamadas programas, o software. El software se divide en dos grandes grupos:

Software del sistema, y Software de aplicaciones.

* El software del sistema es el conjunto de programas indispensables para que la máquina funcione. Estos programas son, básicamente, el Sistema Operativo, los editores de texto, los compiladores de lenguajes de programación y los utilitarios.
  + El sistema operativo dirige las operaciones globales de la computadora, instruye a la computadora para ejecutar otros programas y controla el almacenamiento y recuperación de archivos de discos duros. Gracias al sistema operativo es posible que el programador pueda introducir y grabar nuevos programas, así como instruir a la computadora para que los ejecute. Ejemplos de sistemas operativos son: Microsoft Windows, MS-DOS, OS/2, Mac OS y UNIX.
* Los programas que realizan tareas concretas (por ejemplo facturación, contabilidad, análisis estadístico, gestión de negocios, etc.) se denominan programas de aplicación.

**Lenguajes de Programación**

Los lenguajes utilizados para escribir programas de computadoras que puedan ser entendidos por ellas se denominan Lenguajes de Programación. Éstos se clasifican en tres grandes categorías: máquina, bajo nivel (ensamblador, o assembler) y alto nivel. Los lenguajes de máquina son aquellos cuyas instrucciones son directamente entendibles por la computadora, y no necesitan traducción posterior para que el CPU pueda comprender y ejecutar el programa. La programación en lenguaje de máquina es difícil, ya que implica escribir directamente en un sistema binario (ceros y unos), por eso se necesitan lenguajes que permitan simplificar ese proceso. Los lenguajes de bajo nivel fueron diseñados con ese fin. Éstos son dependientes de la arquitectura física de la computadora y de un conjunto específico de instrucciones para el CPU, y los programas escritos en ellos deben ser traducidos a lenguaje máquina para poder ser ejecutados. Un lenguaje típico de bajo nivel es el lenguaje ensamblador (Assembler).

Los lenguajes de programación de alto nivel son aquellos en los que las instrucciones o sentencias a la computadora son escritas con palabras similares a los lenguajes humanos – en general lenguaje inglés – lo que facilita la escritura y comprensión por parte del programador. Una propiedad de los lenguajes de alto nivel es que son independientes de la máquina, esto es, las sentencias del programa no dependen del diseño de hardware de una computadora específica. Los programas escritos en lenguajes de alto nivel, al igual que los escritos en lenguajes de bajo nivel, no son entendibles directamente por la máquina, sino que necesitan ser traducidos a instrucciones en lenguaje máquina que entiendan las computadoras. Los programas que realizan esta traducción se llaman Compiladores, y los programas escritos en un lenguaje de alto nivel se llaman Programas Fuente. El compilador traduce el Programa Fuente en un programa llamado Programa Objeto. El proceso de traducción de un programa fuente a un programa objeto se denomina Compilación.

Ejemplos de lenguajes de programación de alto nivel son: Basic, C, Pascal, C++, Cobol, Fortran, C#, Java y Visual Basic.NET.

Existen diversos tipos de lenguajes de programación de alto nivel, según su evolución temporal y el uso que se les quiera dar. Algunos ejemplos de tipos de lenguajes de alto nivel son:

* Estructurados (Basic, C, Pascal)
* Orientados a Objetos (C#,Visual Basic.NET, C++, Java)
* Declarativos (Lisp, Prolog)
* Funcionales (AML, CAML)

**Resolución de problemas con computadora**

El proceso de resolución de un problema con una computadora conduce a la escritura de un programa, y a su ejecución en la misma. Aunque el proceso de diseñar programas es, esencialmente, un proceso creativo, se pueden considerar una serie de fases o pasos comunes que generalmente deben seguir todos los programadores. Las fases de resolución de un problema con computadoras son:

* Análisis del problema
* Diseño del algoritmo solución
* Codificación
* Compilación y Ejecución
* Verificación
* Depuración
* Documentación

Las dos primeras fases conducen a un diseño detallado escrito en forma de algoritmo. Un algoritmo es un método para resolver un problema mediante una serie de pasos (instrucciones) precisos, definidos y finitos. Durante la tercer etapa (codificación) se implementa (poner en funcionamiento) el algoritmo en un código escrito en un lenguaje de programación, reflejando las ideas desarrolladas en la fase de análisis y diseño.

La fase de compilación y ejecución traduce y ejecuta el programa. En las fases de verificación y depuración el programador busca errores de las etapas anteriores y los elimina. Cuanto más tiempo se invierta en la fase de análisis y diseño, menor será el tiempo necesario para depurar el programa. Por último, debe realizarse la documentación del mismo.

Ejemplos de algoritmos son: instrucciones para andar en bicicleta, hacer una receta de cocina, obtener el máximo común divisor de dos números, etc. Los algoritmos se pueden expresar en fórmulas, diagramas de flujo y pseudocódigos.

**Entorno de Programación**

Un entorno de programación es un programa que contiene, además del compilador, utilitarios y herramientas. Estos elementos están integrados, de modo que pueden llamarse fácilmente unos a otros durante el proceso de programación. Por este motivo, a los entornos de programación frecuentemente se los identifica con la sigla IDE (en inglés, Integrated Development Envirnonment, o Entorno de programación integrado).

Un entorno de programación típico contiene:

* Un editor, que proporciona el medio para introducir el texto y los símbolos que constituyen el código fuente
* Un intérprete o un compilador, que convierte el código fuente en instrucciones que la computadora puede comprender y ejecutar.
* Un depurador (o debugger), que ayuda a analizar y corregir errores en tiempo de ejecución.
* Ayuda (manuales, tutoriales, ejemplos, etc.) integrada

El proceso de diseño, codificación, depuración y ejecución de un programa es mucho más fácil y rápido cuando se utiliza un buen entorno de programación.

Un ejemplo de entorno de programación es Microsoft Visual Studio 2005.

**Tipos de Datos**

Las diferencias piezas de información con las que un programa trabaja se conocen colectivamente como “datos”. Todos los datos tienen un “tipo” asociado con ellos, que determina la naturaleza del conjunto de valores que aquel puede tomar. Por ejemplo, un dato puede ser un simple caracter, tal como ‘B’, un valor entero como 35, un número real como 1,4142 o una cadena de caracteres como “Hola Mundo”, entre otras cosas. Una operación de suma no tiene sentido con caracteres de texto, sólo con números. Por consiguiente, si el compilador detecta una operación de suma de dos caracteres normalmente producirá un error. Incluso entre tipos numéricos la operación de suma se almacena de modo distinto, ya que los números enteros y los reales se almacenan de formas distintas en memoria. A menos que el programa conozca los tipos de datos no puede ejecutar correctamente la operación de suma.

La asignación de tipos a los datos tiene dos objetivos principales:

1- Detectar errores de operaciones en los programas durante la fase de codificación.

2- Determinar cómo se ejecutarán las operaciones entre datos.

A los lenguajes que exigen que todos los datos utilizados deban tener sus tipos declarados explícitamente se los conoce como “fuertemente tipados”.

El tipo de un dato puede ser convertido bajo ciertas condiciones a otro tipo. Este mecanismo explícito de conversión de tipos de datos se suele denominar “CAST”.

**Variables y Constantes**

Todos los programas necesitan almacenar datos temporalmente para poder procesarlos y generar asi la salida esperada. Estos datos, a grandes rasgos, pueden clasificarse en dos grupos:

1 - A un dato cuyo valor no puede cambiar durante la ejecución de un programa se lo denomina *Constante*. Las constantes deben ser declaradas antes de su utilización y se deben tener un valor ya asignado al momento de la compilación del programa.

2 - Por su parte, los datos de un programa cuyo valor puede cambiar durante la ejecución del mismo se conocen como *Variables*. Una variable es, en realidad, una posición de memoria con nombre (*Nombre de la Variable*), y que contiene un valor (*Valor de la Variable*). Las variables se asemejan a cajas o buzones, donde cada una de las cuales tiene un número y contiene un valor. Existen tantos tipos de variables como tipos de datos diferentes.

**Sentencias**

Las sentencias describen acciones algorítmicas que pueden ser ejecutadas. En general, las sentencias se clasifican en *ejecutables* (especifican, por ejemplo, operaciones de cálculos aritméticos y entradas/salidas de datos) y *no ejecutables* (no realizan acciones concretas ni afectan a la ejecución del programa, sino que ayudan a su legibilidad).

Cada sentencia ejecutable se traduce por el compilador en una o más instrucciones de lenguaje máquina, que se copian en el archivo objeto y posteriormente se ejecutan. Las declaraciones, por su parte, describen el propósito y el significado de cada identificador definido por el programador; no se traducen a instrucciones en lenguaje máquina y no aparecen en el archivo objeto.

Las sentencias se clasifican, según su tipo y número, en:

* Sentencias Simples: son sentencias que no contiene ninguna otra sentencia. El ejemplo más típico de sentencia simple es la *sentencia de asignación*, la cual se utiliza para almacenar un valor en una variable. La operación de asignación se suele representar en pseudocódigo con el símbolo ‘🡨’, para denotar que el valor situado a su derecha se almacena en la variable situada a la izquierda:
  + *Variable* 🡨 *Valor*
  + En la mayoría de los lenguajes, el operador 🡨 se sustituye por el “=”. Ej.: a = 2. (ATENCIÓN: la asignación sólo será válida si el valor es válido para el tipo de dato definido para la variable).
* *Sentencias estructuradas*: son sentencias compuestas de otras sentencias que se ejecutan en secuencia, condicionalmente o repetidamente.

**Operadores y Expresiones**

Las variables y constantes se pueden procesar utilizando operaciones y funciones adecuadas para sus tipos.

Se denomina *expresión* a un conjunto de variables y / o constantes unidas por operadores.

Si en una expresión existe más de una operación debe tenerse en cuenta que existen una serie de reglas para definir la prioridad en la que éstas se realizarán. Por este motivo es que se suelen utilizar los paréntesis para establecer la prioridad de aplicación de los operandos.

Existen diversos tipos de operadores, por ejemplo:

* Aritméticos: son apropiados únicamente para tipos numéricos. Ejemplos de operadores aritméticos son “ + ”, “ - ”, “ \* ” y “ / ”, los cuales permiten obtener el resultado de la suma, la resta, la multiplicación y la división de dos datos respectivamente.
* De relación: los operadores de relación ( o relacionales) se utilizan para expresar condiciones y describen una relación entre dos valores. Ejemplos de operadores relacionales son “ < ” (Menor que), “ >” (Mayor que), “ = ” (Igual a) y “ <> ” (Distinto a).

Los operadores aritméticos y los relacionales se utilizan de la siguiente forma: *variable o constante* ***operador*** *variable o constante*. Por ejemplo: a + b, c/d, a<b, c<>d, etc.

Lógicos: estos operadores se utilizan con constantes lógicas de forma similar al modo en que los operadores aritméticos se utilizan con las constantes numéricas. Estos operadores trabajan con operandos que son expresiones lógicas. La operación *and* (y) combina dos condiciones simples y produce un resultado verdadero sólo si los dos operandos son verdaderos. La operación *or* (o) es verdadera si uno de los dos operandos es verdadero. La operación *not* (no) actúa sobre una sola condición simple u operando y simplemente niega (o invierte) su valor. Existen otros operadores lógicos además de los mencionados.

**Estructuras de Control Repetitivas**

El concepto de flujo de control a través de un programa se refiere al orden en que se ejecutan las acciones individuales de un programa. Aunque un flujo normal de un programa estructurado es lineal, existen métodos que permiten salir del flujo lineal a través del uso de las llamadas *estructuras de control*.

Las estructuras de control de un lenguaje de programación son métodos para especificar el orden en que las instrucciones de un algoritmo se ejecutarán. Estas estructuras son, por consiguiente, fundamentales en los lenguajes de programación y en los diseños de los algoritmos.

Existen dos tipos básicos de estructuras de control, que se explicarán a continuación:

* De Selección
* De Repetición o Iteración

**Estructuras de Control Selectivas**

Con frecuencia nos enfrentamos a situaciones en las que se deben proporcionar instrucciones alternativas que pueden o no ejecutarse dependiendo de los datos de entrada, reflejando el cumplimiento o no de una determinada condición. La realización de acciones alternativas o decisiones se especifican utilizando condiciones que son verdaderas o falsas. Estas condiciones se llaman *expresiones lógicas*, o booleanas. Dado que las expresiones lógicas toman el valor verdadero o falso, se necesita una sentencia de control que dirija a la computadora a ejecutar una sentencia si la expresión es verdadera, y otra sentencia en caso de que sea falsa. En la mayoría de los lenguajes de programación de alto nivel esta evaluación se realiza mediante el uso de la estructura de control “***if*** ”. En la figura podemos ver el pseudocódigo de la estructura if, que funciona de la siguiente manera:

1 – Se evalúa la expresión lógica

2 – Si la expresión toma el valor *true* (verdadero), se ejecutará la sentencia A y el control pasará a la sentencia inmediatamente siguiente.

3- Si la expresión toma el valor *false* (falso), entonces sólo se ejecutará la sentencia B y el control pasa de nuevo inmediatamente a la siguiente sentencia del programa. La cláusula *sino* es optativa.

La sentencia *Case* se utiliza para elegir entre diferentes alternativas. Una sentencia case se compone de varias sentencias simples. Cuando *case* se ejecuta, una (y sólo una) de las sentencias simples se selecciona y ejecuta.

Reglas:

* La expresión *selector* se evalúa y se compara con las listas de constantes; las listas de constantes son listas de uno o más posibles valores de selector separados por comas. Sólo se ejecuta una sentencia.
* La cláusula *sino* es opcional, como en la sentencia if.
* Si el valor del selector no está comprendido en ninguna lista de constantes y no existe la cláusula *sino*, nada sucede y el programa sigue su flujo normal. Si el valor del selector no coincide con alguna constante se ejecutan las sentencias a continuación de la cláusula *else*.
* El selector debe ser un tipo ordinal (número entero, caracter, o booleano). Los números reales no pueden ser utilizados ya que no son ordinales.
* Todas las constantes *Case*  deben ser únicas y de un tipo ordinal compatible con el tipo de selector.

**Procedimientos y Funciones**

Una estrategia para la resolución de problemas complejos con computadoras es la división o descomposición del problema en otros problemas más pequeños y fáciles de resolver. Estos subproblemas se implementan mediante módulos o subprogramas. Los subprogramas son una herramienta importante para el desarrollo de algoritmos y programas, de modo que normalmente un proyecto de programación se compone de un programa principal y un conjunto de subprogramas, con las llamadas a los mismos dentro del programa principal. Un subprograma realiza una tarea concreta que se describe con una serie de instrucciones y que, idealmente, debería ser independiente de otros subprogramas.

Los subprogramas de clasifican en *procedimientos* y *funciones*, que son unidades de programas diseñados para ejecutar una tarea específica.

Las funciones normalmente devuelven un solo valor a la unidad de programa (programa principal u otro subprograma) que los referencia. Los procedimientos pueden devolver cero, uno o varios valores.

Ambos están compuestos por un grupo de sentencias a las que se asigna un nombre (identificador) y constituyen una unidad de programa a la que se puede invocar desde el programa principal u otra función o procedimiento.

**Visibilidad de Variables**

Las variables que intervienen en un programa con procedimientos pueden ser de dos tipos: variables locales y variables globales.

Una variable local es una variable que está declarada dentro de un subprograma, y se dice que es local al subprograma. Una variable local sólo está disponible durante el funcionamiento del mismo, y su valor se pierde una vez que el subprograma termina.

Las variables declaradas en el programa principal se denominan variables globales. A diferencia de las variables locales, cuyos valores se pueden utilizar solo dentro del subprograma en el que están declaradas, las variables globales pueden ser utilizadas en el programa principal y en todos los subprogramas.

Hay que tener especial precaución al trabajar con variables globales, ya que al ser recursos compartidos todos los subprogramas pueden tener acceso simultáneo a ellas y se pueden producir errores lógicos debidos a la concurrencia. Por lo general, es una buena práctica evitar el uso de variables globales desde subprogramas a menos que sea estrictamente necesario.

Las variables también pueden ser declaradas dentro de un bloque o estructura de control, y se comportarán como variables locales únicamente dentro de dicho bloque o estructura.

**Bibliotecas**

En la mayoría de los lenguajes de programación de alto nivel es posible guardar un subprograma o un conjunto de subprogramas como un archivo independiente al cual se denomina genéricamente *biblioteca* (del inglés *library*). Las bibliotecas pueden luego ser referenciadas dentro de múltiples programas para que éstos puedan hacer uso de los subprogramas incluidos en aquella.

La salida del proceso de compilación de un código fuente se denomina *Programa* *Objeto*, que es ni más ni menos que el código de máquina generado a partir del código fuente (SIN INCLUIR EL CODIGO DE LAS BIBLIOTECAS UTILIZADAS). El *Programa* *Objeto* es luego utilizado como entrada para que otro componente de software, llamado *linker,* lo asocie con los subprogramas que se e

**Arrays**

Una estructura de datos es una colección de datos organizados de un modo particular. Las estructuras de datos pueden ser de dos tipos:

* Estructuras de datos estáticas: son aquellas en las que se asigna una cantidad fija de memoria cuando se declara la variable.
* Estructuras de datos dinámicas: son aquellas cuya ocupación de memoria puede aumentar o disminuir en tiempo de ejecución.

Un array (o arreglo) es una estructura de datos en la que se almacena una colección de datos del mismo tipo (por ejemplo, los sueldos de los empleados de una empresa). Dicho de otra forma, un array es una lista de un número finito *n* de elementos del mismo tipo que se caracteriza por:

* + Almacenar sus elementos en posiciones de memoria contiguas
  + Tener un único nombre de variable (por ejemplo *salarios*) que representa a todos los elementos
  + Permitir acceso directo o aleatorio a sus elementos individuales

Los arrays se clasifican en unidimensionales y multidimensionales.

Un array de una dimensión (también conocido como vector) es un tipo de datos estructurado compuesto de un número de elementos finitos, tamaño fijo y elementos homogéneos. *Finitos* indica que hay un último elemento, *tamaño fijo* significa que el tamaño del array debe ser conocido en tiempo de compilación y *homogéneo* significa que todos sus elementos son del mismo tipo.

Los elementos del array se almacenan en posiciones contiguas de memoria, a cada una de las cuales se puede acceder directamente mediante un número entero denominado *índice del array*, que identifica la posición del elemento dentro del conjunto.

* Ejemplo:
  + Salarios 🡪 Nombre del Array
  + Salarios[1] 🡪 primer elemento del array

**El estilo de Programación**

El buen estilo de programación es, sin lugar a dudas, una de las características más importantes que debe tener un buen programador. Un programa con buen estilo es más fácil de leer, de corregir y de mantener. Aunque la experiencia proporciona el estilo, existen una serie de reglas que se recomiendan seguir desde el principio del aprendizaje de la programación.

* Comentarios: la legibilidad de los programas aumenta considerablemente utilizando comentarios. Un comentario es un texto explicativo más o menos largo, situado en el programa e ignorado por el compilador. Los comentarios son considerados parte del código fuente por más que sean ignorados en las etapas de compilación y ejecución, y tienen importancia primordial en las fases de análisis, puesta a punto y mantenimiento. Los comentarios son una parte fundamental de la documentación de un programa, ya que ayudan al programador y a otras personas a la comprensión del mismo. No es raro encontrar programas en los cuales los comentarios ocupan más lugar, incluso, que las propias instrucciones.
* Elección de nombres significativos para variables y procedimientos: las variables, constantes, nombres de subprogramas y nombres de programas deben ser significativos para orientar al usuario o a otros programadores sobre lo que representan: X, AS, JJ no son identificadores significativos.
* Identación: aunque no suele ser requerido por los compiladores de los lenguajes de alto nivel, es una práctica habitual realizar una identación (o tabulación) en cada uno de los bloques y unidades de programas fundamentales para mejorar la legibilidad del código fuente.
* Espacios y Líneas en Blanco: con el mismo fin de mejorar le legibilidad y comprensibilidad del programa es recomendado utilizar espacios en blanco antes y después de un operador, así como también dejar una línea en blanco entre partes importantes o que estén lógicamente separadas.
* Validación: los programas no pueden considerarse correctos hasta que han sido validados utilizando un rango amplio de datos de prueba para contemplar todas las posibles direcciones que el flujo de control puede tomar en tiempo de ejecución.