UNIVERSIDAD AUTÓNOMA DE NUEVO LEON
FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA
DIVISIÓN DE ESTUDIOS DE POSTGRADO

ANTOLOGÍA DE LENGUAJES DE PROGRAMACIÓN

POR

ADRIANA RAMÍREZ HERNÁNDEZ

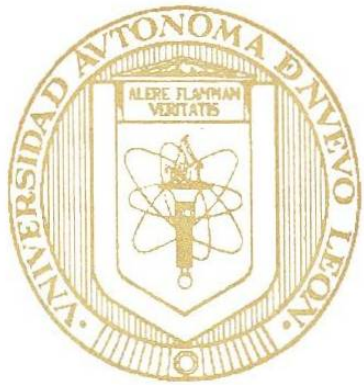
TESIS
EN OPCIÓN AL GRADO DE MAESTRO EN CIENCIAS DE LA ADMINISTRACIÓN CON ESPECIALIDAD EN SISTEMAS

SAN NICOLÁS DE LOS GARZA, N.L. FEBRERO DEL 2000

ARH

ANTOLOGÍA DE LENGUAJES DE PROGRAMACIÓN

UNIVERSIDAD AUTÓNOMA DE NUEVO LEON
FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA
DIVISIÓN DE ESTUDIOS DE POSTGRADO



ANTOLOGÍA DE LENGUAJES DE PROGRAMACIÓN

POR

ADRIANA RAMÍREZ HERNÁNDEZ

T E S I S

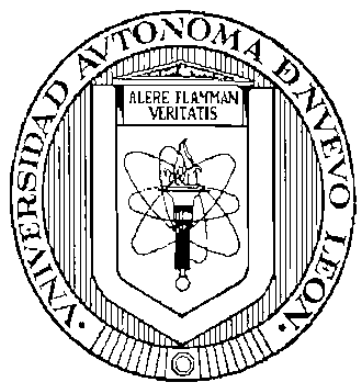
EN OPCIÓN AL GRADO DE MAESTRO EN CIENCIAS DE LA ADMINISTRACIÓN CON ESPECIALIDAD EN SISTEMAS

SAN NICOLÁS DE LOS GARZA, N.L. FEBRERO DEL 2000

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

Facultad de Ingeniería Mecánica y Eléctrica

DIVISIÓN DE ESTUDIOS DE POST-GRADO



ANTOLOGÍA DE LENGUAJES DE PROGRAMACIÓN

POR

ADRIANA RAMÍREZ HERNÁNDEZ

TESIS

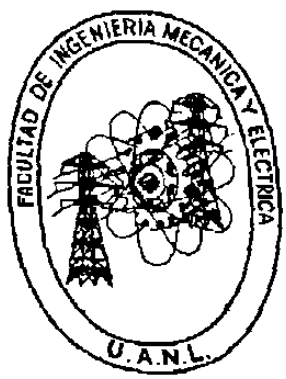
EN OPCIÓN AL GRADO DE MAESTRO EN CIENCIAS DE LA ADMINISTRACIÓN CON ESPECIALIDAD EN SISTEMAS

SAN NICOLÁS DE LOS GARZA, N.L., A FEBRERO DEL 2000

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

Facultad de Ingeniería Mecánica y Eléctrica

DIVISIÓN DE ESTUDIOS DE POST–GRADO



ANTOLOGÍA DE LENGUAJES DE PROGRAMACIÓN

POR

ADRIANA RAMÍREZ HERNÁNDEZ

TESIS

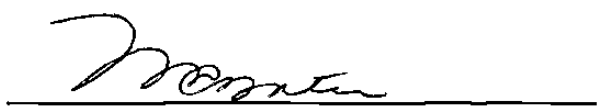
EN OPCIÓN AL GRADO DE MAESTRO EN CIENCIAS DE LA ADMINISTRACIÓN CON ESPECIALIDAD EN SISTEMAS

SAN NICOLÁS DE LOS GARZA, N.L., A FEBRERO DEL 2000

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN
FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA
DIVISIÓN DE ESTUDIOS DE POST-GRADO

Los miembros del comité de tesis recomendamos que la tesis "Antología de Lenguajes de Programación" realizada por la alumna Adriana Ramírez Hernández, matrícula 918241 sea aceptada para su defensa como opción al grado de Maestro en Ciencias de la Administración con especialidad en Sistemas.

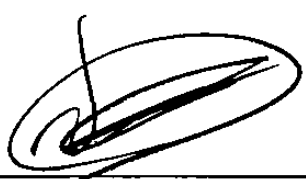
El comité de Tesis



Asesor
M.A. Matías Alfonso Botello Treviño

Coasesor
M.C. Cástulo E. Vela Villarreal

Coasesor
M.C. Roberto Villarreal Garza



Vo. Bo.
M.C. Roberto Villarreal Garza
División de Estudios de Post-grado

San Nicolás de los Garza, N.L., a Febrero del 2000

TABLA DE CONTENIDO

AGRADECIMIENTOS

A Dios por haberme dado vida y ayudado en la conclusión de mi post-grado.

A mi padre y madre por su amor y confianza que me tienen en todo proyecto que inicio. Y por celebrar conmigo mis triunfos y alentarme en mis fracasos.

A mis hermanos Miguel y Rosy porque me hacen sentir que soy importante en sus vidas.

A mis maestros por compartir su gran sabiduría conmigo.

A mi asesor y coasesores por atenderme en cualquier momento en que los he necesitado, les quedo profundamente agradecida.

SINTESIS

En el presente documento, se busca concientizar el porqué estudiar lenguajes de programación, como han surgido factores para cambio y que características lo hacen para poder definir que tan bueno es un lenguaje.

Además se recuerdan las etapas que atraviesa un compilador para reconocer cada uno de los elementos de un programa o de un lenguaje.

También, se define de una manera formal la sintaxis de un programa mediante la gramática BNF .

Otros de los aspectos son reconocer y recordar algunos de los datos más importantes que generalmente mantiene un lenguaje de programación incluyendo la definición e implementación en memoria.

También abarca de una manera sencilla el funcionamiento y la implementación en memoria de diversos subprogramas como corrutinas, recursivos, entre otros. Otros de los temas importantes que se explican es el manejo de proposiciones y expresiones que tiene un programa y qué significado tiene un subprograma o una variable dentro de un subprograma o programa principal e indicar y explicar los diferentes mecanismos para transmitir datos mediante diferentes alternativas.

INTRODUCCION

La Escuela juega un papel importante en el desarrollo integral del alumno, y se va adaptando al mundo en el que vivimos, el cual se encuentra en constante proceso de cambio y transformación.

Un lenguaje de programación podríamos decir que es cualquier notación para la descripción de algoritmos y estructuras de datos.

Un elemento básico que resulta fundamental en la programación es el uso de un lenguaje de programación y una computadora, es por ello necesario que se conozca y se comprenda las características de un lenguaje expresadas mediante líneas de codificación. También resulta de vital importancia para el desarrollo de un lenguaje el conocimiento de implementaciones en memoria de algunos elementos de un programa del lenguaje.

En este trabajo se incluyen aspectos, tales como: un panorama general a la introducción de lenguajes de programación, esquemas de traducción de lenguajes de programación, tipos elementales de estructuras y abstractos, control de secuencias, control de datos y administración de la memoria.

El docente y el alumno son factores imprescindibles en el desarrollo de este trabajo logrando así servir de base para contribuir de manera flexible dentro del proceso educativo.

OBJETIVOS DE LA TESIS

- Proporcionar al docente una antología de apoyo para facilitar la enseñanza.
- Proporcionar al educando una antología como material de referencia bibliográfica de consulta.
- Proporcionar al docente la antología para que la analice a fondo los temas de cada unidad y así mismo que la complemente con temas opcionales a medida que pasa el tiempo.
- Propiciar en el alumno la concentización y la valorización la antología como un recurso informativo.
- Fomentar un ambiente de confianza y seguridad durante el desarrollo de actividades.

PLANTEAMIENTO DEL PROBLEMA

La ciencia conjuntamente con la educación están cambiando constantemente y a ello la carrera de Licenciatura de Informática en el Instituto Tecnológico de Estudios Superiores de la Región Carbonífera ubicado en la Ciudad de Agujita, Coahuila México, considerando a qué este desarrollo evoluciona acorde a la tecnología que se va presentando. Por ello se necesita que el alumno tenga algunos conocimientos básicos y que el mismo este preparado cada día más.

La antología que propongo a desarrollar de apoyo a la asignatura de Lenguajes de Programación de la carrera de Licenciatura de Informática para el docente facilitando el proceso de aprendizaje así elevar el nivel académico y como de apoyo de consulta para el alumno. La mayoría de los temas que contiene la misma presenta un análisis de todos los conocimientos adquiridos por los alumnos durante el transcurso de su carrera y algunos que conocerán.

JUSTIFICACION

La carrera de Licenciatura en Informática es de suma importancia dentro del campo laboral de la actualidad, retomando lo que anteriormente maneje resulta benéfico que el alumno posea mayor nivel educativo con el apoyo brindado por el docente y pueda afrontar el quehacer diario realizado.

Propongo realizar una antología de apoyo donde contenga los temas teóricos de la asignatura Lenguajes de Programación, los cuales abordan de la diferente bibliografía existente y así como darle la oportunidad al docente de que analice profundamente cada tema de los capítulos comprendidos y al mismo tiempo complementará la misma con temas de la actualidad a medida que pasa el tiempo.

HIPOTESIS

Facilitar la enseñanza al alumno-maestro y elevar el nivel académico de la Institución.

1. INTRODUCCION A LOS LENGUAJES DE PROGRAMACION

El objetivo de este capítulo es dar a conocer la evaluación de los lenguajes de programación así como su clasificación.

1.1. Porque estudiar lenguajes de programación.

Debido a que muchos programadores en la práctica trabajan en instalaciones de computadoras donde se requiere el uso de cierto lenguaje en particular, surge la siguiente pregunta:

¿ Cuál es la ventaja de estudiar una variedad de lenguajes que es improbable que usemos?

La podemos contestar con las siguientes justificaciones:

1. Mejora el conocimiento del lenguaje que se esta usando.
2. Enriquece su vocabulario de construcciones útiles de programación.
3. Permite una mejor selección del lenguaje de programación.
4. Hace más fácil el aprendizaje de un nuevo lenguaje.
5. Facilita el diseño de un nuevo lenguaje de programación.

1.2. Historia de los lenguajes de programación.

El programador, diseñador e implementador de un lenguaje de programación deben comprender la evolución histórica de los lenguajes para poder apreciar lo siguiente:

- Permita ver la evolución de familias de lenguajes de programación.
- Ver la influencia que ejercen las arquitecturas y aplicaciones de las computadoras sobre el diseño de lenguajes.
- Evitar futuros defectos de diseño aprendiendo las lecciones del pasado.

A continuación se presentan algunos de los factores importantes que han ocasionado el seguimiento de nuevos lenguajes y se hace mención de algunos de ellos.

Año 1950 a 1955.

Factores: Computadoras primitivas, programación en lenguaje ensamblador y de máquina.

Lenguaje: Lenguaje ensamblador y lenguaje de alto nivel en prueba pero ninguno en uso.

Año 1956 a 1960.

Factores: Computadoras pequeñas, lentas y caras. Sistema de almacenamiento en cinta magnética, compiladores, intérpretes de software, optimización de código, manejo de almacenamiento dinámico y procesamiento de listas.

Lenguajes: FORTRAN, ALGOL 58, ALGOL 60, COBOL y LISP.

Año 1961 a 1965.

Factores: Computadoras grandes y caras. Sistema de almacenamiento en cinta magnética, multiprogramación y se proponen como meta generar lenguajes de propósito general.

Lenguajes: FORTRAN IV, SNOBOL, APL(no implementado), ALGOL 60, COBOL 61(extendido)

Año 1966 a 1970.

Factores: Computadoras compatibles con conjuntos de instrucciones, costo variable, sistema de almacenamiento masivo y grande, sistema operativo interactivo de tiempo compartido y optimizador de compiladores.

Lenguajes: PL/1, FORTRAN 66, SNOBOL4, APL360, BASIC SIMULA 67 y COBOL 65(estándar)

Año 1971 a 1975.

Factores: Microcomputadoras, sistema de almacenamiento masivo grande, programación estructurada, ingeniería de software, simplicidad como meta en el diseño de lenguaje.

Lenguajes: PASCAL, COBOL 74(estándar) y PL/1(estándar)

Año 1976 a 1989.

Factores: Computadoras potentes y baratas, sistema de almacenamiento masivo, grande y barato, programación estructurada, ingeniería de software, simplicidad como meta en el diseño de leguajes distribuidos de computación, modelo de cliente / servidor, programación concurrente y orientada a objetos, y confiabilidad.

Lenguajes: C, C++ y SMALLTALK.

Año 1989 a la fecha.

Factores: Computadoras potentes y baratas, sistema de almacenamiento masivo y grande y barato, el diseño de lenguajes de consulta donde el programador solo necesita especificar "que hacer" y no "como hacerlo", cada vez la programación es más fácil de entender.

Lenguajes: Access, Paradox, Oarcle.

1.3. Clasificación.

Para ver la clasificación de los lenguajes en procedural, funcional, lógico y orientado a objetos lo apreciamos con la siguiente tabla adjunto con ciertas características a analizar.

Modelo de lenguaje	Procedural	Funcional	Lógico	Orientado a objetos
Tipo de datos	Arreglos y registros	Simbólico, numérico y listas	Simbólico, numérico, listas y predicado	Simbólico, numérico y manejo de clases
Manipulación de datos	Asignación y funciones	Funciones	Variables	Asignación de valores en objetos
Programa de control	Secuencias, ciclos y recursión	Función, ciclos y recursión	Patrones de búsqueda y recursión	Traslado de mensajes
Estructura del programa	Por bloques ambiente global	Funciones en hechos	Reglas y clases	Objetos en jerarquía
Modelo de iteración	Compilador	Compilador	Compilador	Compilador

1.4. Características de un buen lenguaje.

Las características de un buen lenguaje nos sirven para evaluar un lenguaje de programación, con diferentes puntos de vista del programador, así como del diseñador del lenguaje y del implementador del lenguaje que a continuación se listan:

1. Expresividad.

Se refiere al ingenio de un lenguaje para reflejar claramente el significado deseado por el diseñador del algoritmo (programador) es decir con conceptos sencíllos, simples y unificados.

2. Bien definido.

Relata que la sintaxis y semántica del lenguaje no contiene ambigüedad es internamente consistente y completo por consiguiente el programador debe poder predecir exactamente el comportamiento de cada expresión antes de que realmente se ejecute.

3. Tipos y estructuras de datos.

Narra la capacidad de un lenguaje para soportar una variedad de valores de datos(enteros, reales, cadenas, punteros, entre otros) y colecciones de valores de los datos anteriores vistas como un solo dato conocidas como estructuras incluyendo estructuras dinámicas.

4. Modularidad.

Se refiere a la aptitud de definir procedimientos y funciones independientes y comunicarlos vía parámetros o variables locales como el programa llamador.

5. Facilidades de entrada-salida.

Se refiere a como soporta el lenguaje los archivos de acceso secuencial, indexado y directo, así como las funciones sobre bases de datos y recuperación de información.

6. Transportable.

Se refiere al lenguaje que esta implementando en diferentes computadoras.

7. Eficiente.

Es aquel que permite una compilación y ejecución rápida sobre las máquinas que esta implementado.

8. Pedagogía.

Se refiere a los lenguajes de programación que son fáciles de enseñar y de aprender.

9. Generalidad.

Se refiere a que tan útil sea el lenguaje en un amplio rango de aplicaciones.

2. ESQUEMAS DE TRADUCCION DE LOS LENGUAJES DE PROGRAMACION

El objetivo de este capítulo es dar a conocer la estructura de un programa, así como el contenido del mismo y los pasos que atraviesa un traductor para reconocer todos los elementos de un programa y así ejecutarlo.

2.1 Definiciones importantes.

A continuación se presentan algunas definiciones importantes para comprender el objetivo de este capítulo.

Sintaxis de un lenguaje de programación.- Es el conjunto de reglas y criterios de escrituras que permiten la formación de programas correctos en un lenguaje.

Semántica de un lenguaje de programación.- Es el significado de varias construcciones sintácticas bien escritas.

Compilador.- Es un traductor el cual utiliza como lenguaje fuente un lenguaje de programación de alto nivel y produce como lenguaje objeto un lenguaje maquinal listo para ejecutarse. Algunas de las características importantes son: solo se traduce una vez y genera programas rápidos pero es necesario recompilar para modificaciones.

Intérprete.- Es un traductor que utiliza como lenguaje fuente un lenguaje de programación de alto nivel y no genera lenguaje objeto; es decir compila una línea del programa y luego la ejecuta sí esta correcta. Algunas de las características principales son: convierte muchas veces, es lento para ejecutarse.

2.2. Criterios generales de la sintaxis y semántica.

El propósito primario de la sintaxis es proporcionar una notación para comunicar la información entre el programador y el procesador del lenguaje de programación.

Hay muchos criterios para seleccionar una sintaxis de un lenguaje como son los siguientes:

1. Legibilidad.

La legibilidad se realza por tener formatos naturales de proposición, proposiciones estructuradas, identificadores con longitud sin restricciones, provisión para encajar comentarios y declaraciones de datos. Además establece que las construcciones del programa que hacen diferentes parezca diferente.

2. Fácil de escribir.

Las características de la sintaxis que hacen a un programa fácil de escribir se oponen algunas veces a aquellas que lo hacen fácil de leer. Se realza por el uso de estructuras de sintaxis concisas y regulares, establece que se permita que las declaraciones y las operaciones se dejen de especificar esto provoca que los programas sean mas cortos y más fáciles de escribir, pero más difíciles de leer. También establece que no se permita una sintaxis redundante.

3. Falta de ambigüedad.

Cuando se permite dos o más interpretaciones diferentes para una construcción. La definición de un lenguaje proporciona en forma ideal un significado único para cada construcción de sintaxis que un programador pueda escribir.

2.3. Elementos sintácticos de un lenguaje.

El estilo sintáctico de un lenguaje es determinado por la selección de varios elementos sintácticos como son: conjunto de caracteres, identificadores, símbolos, operadores, comandos y palabras reservadas, comentarios, espacios, delimitadores, corchetes, expresiones y proposiciones.

2.4. Etapas de la traducción.

Para la traducción es decir la transformación de un programa fuente escrita en un lenguaje de alto nivel en una secuencia equivalente de instrucciones maquinales se requiere de una serie de etapas.

Cada una de las cuales tiene una finalidad específica y se llevan a cabo en una forma secuencial por medio de etapas la salida de una etapa es la entrada de la siguiente. A cada una de estas etapas se le llama "Fase de un compilador". Las cuales son las siguientes:

Análisis lexicográfico.

Es la rutina de entrada para el traductor que lee las líneas sucesivas del programa de entrada, analizando carácter por carácter es decir las subdivide en componentes elementales: identificadores, delimitadores, números, espacios en blanco, palabras reservadas y las agrega a la tabla de símbolos.

Análisis sintáctico.

Aquí se detecta las estructuras más grandes del programa, proposiciones, expresiones y declaraciones utilizando los elementos producidos por la etapa anterior verificando que estén agrupados en una secuencia o estructura permitida por las reglas sintácticas del lenguaje a traducir. Utiliza un medio de almacenamiento para colocar todos los elementos anteriores los cuales van a ser procesados por el analizador semántico.

Análisis de semántica.

Aquí las estructuras analizadas anteriormente son reconocidas de acuerdo a un conjunto de reglas que definen el efecto operacional del programa cuando se traduce y se ejecuta.

Optimización.

El objetivo es refinar el código generado para que se mejore el rendimiento en tiempo de ejecución, se intenta localizar las construcciones semánticamente redundantes, el uso ineficiente de registros, sacar las operaciones constantes de los ciclos, etc. para que el programa ocupe menos espacio en memoria. Generalmente está fase ocurre antes y después de la etapa de generación de código.

Generación de código intermedio.

Crea una secuencia de instrucciones en un lenguaje máquina que remplaza al programa fuente.

2.5. Definición formal de la sintaxis.

Para definir la sintaxis utilizaremos la gramática de BNF (forma de Backus Naur) que se compone de un conjunto finito de reglas que juntas definen al lenguaje de programación. Nace originalmente por Noam Chomsky la cual fue adaptada posteriormente por Backus y Naur y usada para describir la sintaxis del lenguaje ALGOL 1962. BNF se compone de los siguientes símbolos:

::= significa "está definida como".

{ } Significa "cualquier secuencia de 0 o más de los items cerrados".

[ ] Significa "0 o una ocurrencia de los items encerrados".

| Significa "O" en el sentido exclusivo.

Los elementos que componen las reglas en BNF son nombres de *clases de tokens (tales como "identificador") o símbolos* simples del alfabeto del lenguaje.

Una regla de BNF tiene siempre su nombre de una clase de tokens a su izquierda, seguida de ::=, seguida de una secuencia de nombres de clases de tokens y/o símbolos, separados posteriormente por | y agrupados mediante {} Entonces por ejemplo la siguiente regla BNF define exactamente lo que significa la clase de tokens "letra" es:

Letra::=a|b|c|d|.....|z|A|B|C|.....|Z

En castellano interpretaríamos está definición como una letra se define como "a", o "b", o ....., o "z" o "A", o "B", o ....."Z".

Otro ejemplo que define lo que significa la clase de tokens "dígito" es ::= 1|2|3|4|5|6|7|8|9

De acuerdo a lo anterior podríamos construir una nueva regla llamada "identificador": identificador::= letra {letra|dígito}

Lo que significa: "un identificador se define como una letra, seguida de 0 ó más ocurrencias de una letra o dígito.

2.6. Unión y tiempos de unión.

Definiciones importantes.

Unión.- Es el otorgamiento de una característica a un elemento de todas las existentes.

Tiempo de unión.- Es el tiempo en el cual y durante el cual se da esa característica.

Hay diferentes tiempos de unión que se dan a un elemento de un lenguaje de programación los cuales son los siguientes:

a) Tiempo de ejecución. Ocurre cuando una variable toma un valor puede ser en la entrada a un subprograma o en puntos arbritarios durante la ejecución del programa.

b) Tiempo de definición. Se refiere a las alternativas de estructuras que nos presenta un lenguaje de programación al momento de seleccionar una cuando empezamos a escribir un programa.

c) Tiempo de traducción. Se refiere a cuando el programador escribe su programa, él tiene que seleccionar diferentes estructuras (nombre a una variable, tipo a una variable, etc.) a usar que para el compilador representan uniones durante la traducción o también hay ocasiones que otras uniones sean escogidas por el compilador del lenguaje sin que las proporcione el programador.

3. TIPOS DE DATOS ELEMENTALES, DE ESTRUCTURAS Y ABSTRACTOS

El objetivo de este capítulo es enterarse de los diferentes tipos de datos y estructuras aplicables en diferentes tipos de lenguajes de programación.

3.1 Dato objeto, variables, constantes y declaración.

Definiciones importantes.

Dato objeto.- Es un recipiente que sirve para almacenar información.

Tipo de dato.- Es una clase de dato objeto junto con una serie operaciones para crearlo y manipularlo.

Dato objeto variable.- Es un dato objeto que tiene un nombre y una unión a uno o más valores que pueden cambiar durante todo el tiempo de su vida.

Dato objeto constante.- Es un dato objeto que tiene un nombre y una unión a uno o más valores que no pueden cambiar durante todo el tiempo de su vida.

Existen 2 tipos principales de datos objeto. Los datos objeto elementales que son aquellos que solo permiten almacenar su valor y manipularlo siempre como una unidad y los datos objeto de estructura de datos los cuales están formados por varios datos objeto.

Declaración.- Es una instrucción que sirve para informar al traductor sobre el tipo de datos y los atributos de los datos objeto utilizados. Hay 2 tipos de declaración: explícita la cual la realiza en algún lugar el programador y la implícita es aquella que por definición del lenguaje le da ciertos atributos a un dato objeto cuando no existe por parte del programador.

3.2. Especificaciones de tipos de datos elementales.

Para especificar un tipo de datos elemental se analiza para cada uno lo siguiente:

Atributos. Se refiere a las características que tiene un tipo de datos a analizar.

Valores. Se refiere a todos los valores que se puede tomar un tipo de datos determinado.

Operaciones. Se refiere al conjunto de operaciones que se puede realizar con este tipo, por ejemplo: aritméticas(+, -, *, residuo), relacionales(<, >, >=,<= ,=), funciones estándar definidas por el lenguaje y de asignación.

A continuación se analiza la especificación de algunos tipos de datos elementales.

1. Tipo de dato entero.

Este tipo de dato es él más primitivo su característica principal que su propio tipo de dato, los valores que puede tomar es de acuerdo al lenguaje, la arquitectura de la computadora generalmente se representa mediante un rango de valores que va desde el mínimo hasta un máximo. Las operaciones que se pueden hacer son aritméticas, relacionales, funcionales definidas por propio lenguaje y de asignación.

2. Tipo de dato real de punto fijo.

Estos números reales se especifican como una secuencia de dígitos de longitud fija conteniendo un punto decimal entre dos dígitos, los valores que puede tomar van desde un valor mínimo hasta un valor máximo al igual que los enteros pero ha diferencia de que su secuencia no es uniforme, es decir del número 5.0 al 6.0 hay muchos números entre ambos como 5.1, 5.2, 5.3, etc.

Las operaciones que maneja son aritméticas, relacionales, de asignación y funciones definidas por el propio lenguaje como seno, coseno, tangente entre otras.

3. Tipo de datos real de punto flotante.

La especificación de números reales es su propio tipo de dato. Sin embargo es muy común que en un lenguaje de programación este tipo de datos se maneje con varios niveles de precisión, es decir el programador tendría que especificar que nivel de precisión requiere. Este tipo de dato se diferencia del anterior porque se representa con una mantisa y un exponente. Los valores que puede tomar van desde un valor mínimo hasta un valor máximo al igual que los enteros pero ha diferencia de que su secuencia no es uniforme al igual que el anterior. Las operaciones que maneja son: aritméticas, relacionales, de asignación y funciones estándar definidas por el propio lenguaje como seno, coseno, tangente entre otras.

4. Tipo de dato booleano.

Su característica principal es su propio tipo de dato. Los únicos valores que puede tomar son verdadero o falso. Las operaciones que se pueden hacer son conjunción (AND), disyunción(OR) o negación o complemento(NOT) y la operación de asignación.

5. Tipo de dato enumerativo.

Su principal característica es su propio tipo de datos, se define como una lista ordenada de valores distintos y a cada elemento se le asigna un valor entero consecutivo, con el cual identifica a cada elemento, pero no se considera como un rango. Las operaciones manejadas para este tipo son: relacionales, predecesor (proporcionan elemento anterior de la lista), sucesor(proporciona el elemento siguiente de la lista)

6. Tipo de dato carácter.

Su principal característica es su propio tipo de datos, el valor que puede tomar un carácter es de acuerdo al valor conocido como código que se encuentra representado en una tabla de códigos soportados por el hardware y el sistema operativo como la ASCII, EBCDIC.

Las operaciones que se puedan realizar con un carácter son: aritméticas (algunas de ellas dependen del lenguaje de programación), lógicas y de asignación.

3.3. Implementación de datos elementales.

En esta sección tiene como objetivo mostrar como se representan en memoria los diferentes tipos antes expuestos dependiendo de la arquitectura de la máquina y del lenguaje. También se analiza la representación de las operaciones como una de las siguientes formas:

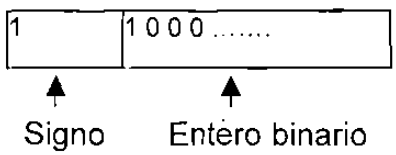
a) Directamente en hardware. Es decir con la ayuda de la arquitectura de la computadora se pueden hacer las diferentes operaciones que se pueden manejar usando los tipos de datos.
b) Como un conjunto de líneas de codificación conocida como función o procedimiento que la ofrece el mismo lenguaje.

A continuación se explica cada uno de ellos.

1. Tipo de dato entero.

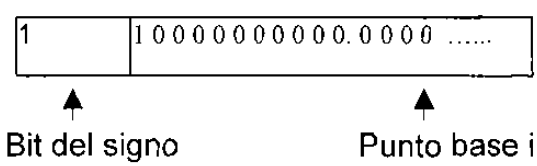
Si su computadora mantiene el registro de 8 bits el valor máximo representable sin signo es del 0 al 255 a el equivalente es binario: 0 a 11111111. Un número con signo en formato positivo no puede ser mayor a 01111111 binario ó 127 decimal, debido a que el siguiente número válido sería 10000000 y al tener encendido el bit más significativo es evidente que representaría un número negativo, que representa al número decimal –128. Si la palabra fuese de 16 bits se aplica en forma parecida al anterior. Para representar las operaciones se auxilia de las que proporciona el mismo hardware.

A continuación se muestra el esquema para representar un número de tipo de dato entero.



2. Tipo de datos real de punto fijo.

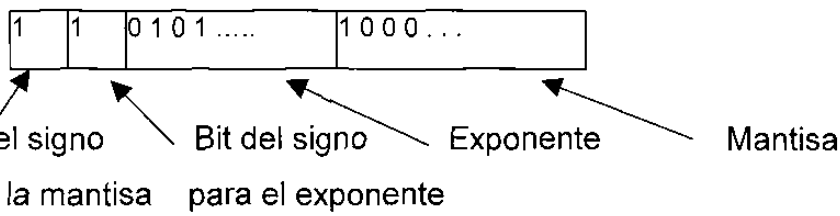
Para representar el número de punto fijo se representa en binario conteniendo un punto entre el mismo, donde una parte significa la entera y la otra representa la parte decimal. Para representar las operaciones se auxilia directamente de las que proporciona el mismo hardware, excepto por las que tiene definidas el propio lenguaje como tangente, coseno, seno, entre otras. Generalmente son simulados por software. A continuación se muestra un esquema para representar un número real de punto fijo.



3. Tipo de dato real de punto flotante.

Para representar el número de punto flotante se representa en binario como en los demás casos ya que la computadora trabaja con números binarios, pero en este caso la idea es representar el número en dos partes su mantisa y exponente. Los números de precisión sencilla generalmente son representados por 24 bits para la mantisa, incluyendo 1 para el bit de signo y 8 para el exponente, incluyendo el signo. Teniendo un punto entre el mismo, donde una parte significa la entera y la otra representa la parte decimal. Los números de precisión doble generalmente son representados por 53 bits para la mantisa, incluyendo 1 para el bit de signo y 11 para el exponente, incluyendo el signo. Para representar las operaciones se auxilia directamente de las que

proporciona el mismo hardware, excepto por las que tiene definidas el propio lenguaje como tangente, coseno, seno, entre otras. Generalmente son simuladas por el software. A continuación se muestra un esquema para representar un número real de punto flotante.



4. Tipo de dato booleano.

Para representar un tipo booleano generalmente usa dos representaciones:

a) Se utiliza un bit particular para el valor (con frecuencia el bit del signo de la representación del número), o con 0 significa que el dato es falso y con un valor de 1 significa que el dato e es verdadero y el resto de la palabra se ignora.
b) Un valor cero en toda la palabra significa que el dato es falso y cualquier otro valor que no sea cero representa el valor positivo. Para representar las operaciones se auxilia directamente de las que proporciona el mismo hardware.

5. Tipo de dato enumerativo.

La representación de almacenamiento para un dato objeto de un tipo de enumeración es directa cada valor en la secuencia de enumeraciones se representa en el tiempo de ejecución por uno de los enteros 0,1,2,3,4, . . . Sin embargo, como se incluyen una pequeña serie de valores y los valores nunca son negativos, generalmente quita el bit del signo de la palabra y deja los bits suficientes para representar el rango de valores requerido. Para implementar las operaciones básicas sobre enumeración es también directa, porque pueden

usarse las operaciones sobre enteros proporcionadas por hardware por ejemplo las operaciones lógicas como: >, <, = pueden implementarse con el uso de operaciones primarias del hardware correspondientes que comparan enteros. Las operaciones de sucesor y predecesor simplemente se requiere sumar o restar uno al entero que representa el valor y verificar que el rango este dentro del adecuado.

6. Tipo de dato carácter.

Los datos carácter son casi siempre soportados directamente por el hardware representados por números binarios.

Por ejemplo si nos basamos a la tabla de código ASCII, cada carácter mantiene un número entero decimal se codifica numéricamente con una combinación de 7 bits puesto que existen un total de $2^7 = 128$ caracteres diferentes.

Por ejemplo para representar la letra A (mayúscula) se representa por el dígito 65 en decimal y en binario 10000001. Para representar las operaciones se auxilia directamente de las que proporciona el mismo hardware, basándose en su orden relativo que tienen en la tabla de códigos.

3.4. Especificaciones de estructuras de datos.

Ahora analizaremos datos objeto cuyos componentes son datos elementales todos del mismo tipo o de diferente a lo que le llamamos datos objeto de tipo de estructura de datos. Para especificar una estructura de datos se analiza lo siguiente:

Atributos. Número, tipo, nombre, número máximo de cada componente.

Operaciones. Operación de selección de componentes, operaciones a estructura completa esto implica a la suma, asignación o resta de los componentes de una estructura con otra sin hacerlo de uno por uno, inserción y supervisión de componentes y creación de componentes, además incluye las operaciones que se pueden hacer con los componentes de la estructura de datos con otros componentes de una estructura de datos e incluso con los mismos componentes de la estructura de datos como aritméticas, relacionales y de asignación.

A continuación se analiza la especificación de algunos tipo de datos.

1. Arreglos o vectores.

Se define como una estructura de datos compuesta por un número fijo de elementos organizados en forma lineal. Los principales atributos son: el tipo de dato, tamaño del vector expresado como el número de componentes, tipo de cada uno en este caso todos son del mismo y el nombre del subíndice, operaciones aritméticas como la suma, resta y de asignación a estructura completa y con los componentes de la estructura se pueden realizar operaciones aritméticas, relacionales y funciones predefinidas por el lenguaje.

2. Registros.

Se define como una estructura de datos compuesta por un número fijo de elementos de diferentes tipos elementales. Los principales atributos son: número de componentes y el tipo de dato de cada componente. Las operaciones que se pueden hacer son: seleccionar un componente mediante el nombre del registro y nombre del elemento a seleccionar, también existen la operación a estructura completa en algunos lenguajes con la restricción que todos sus campos deben de ser del mismo tipo, e incluye todas las operaciones que se pueden hacer con los componentes de la estructura.

3. Cadena de caracteres.

Se define como una estructura de datos compuesta de una secuencia de caracteres. Existen diferentes tipos de cadenas las cuales son las siguientes:

a) Longitud declaradas fijas. La longitud de la cadena será declarada por el usuario. La asignación del nuevo valor a la cadena produce un ajuste a la longitud de la nueva cadena a través del truncamiento de exceso de caracteres o de la adición de caracteres en blanco para producir una cadena de longitud correcta.

b) Longitud variable a un límite declarado. Un dato objeto de la cadena de caracteres puede tener una longitud máxima que se declara en el programa como en el anterior, pero el valor real almacenado en el dato objeto puede ser una cadena de longitud mas corta, posiblemente una cadena de longitud vacía, pero si se le añaden caracteres serán truncados si exceden de la longitud.

c) Longitud ilimitada.

Se refiere a que la longitud de la cadena puede variar en forma dinámica durante la ejecución sin límites. Las operaciones que se pueden realizar con la cadena de caracteres son: concatenación (Implica unir dos cadenas de caracteres para hacer una cadena mas larga), operaciones lógicas con cadenas(>,<,=,>=,<=), selección de subcadenas utilizando subíndices de posición de caracteres y selección de cadenas utilizando patrón de emparejamiento y algunas funciones para manipularlas por ejemplo: Convertirlas a mayúsculas o minúsculas, determinar si la cadena contiene números, caracteres alfabéticos o alfanuméricos.

4. Datos objeto y apuntadores dinámicos.

Un dato objeto apuntador almacena la dirección o localidad de un determinado tipo de dato. Las operaciones que se pueden hacer son:

a) *Creación. Implica proporcionar un bloque de almacenamiento para un dato objeto de una manera dinámica en cualquier instante durante el tiempo de ejecución.*
b) Selección. Implica seguir el valor del indicador para alcanzar el dato objeto designado.
c) Eliminar. Significa liberar el espacio de almacenamiento creado en algún momento durante la ejecución del programa.

5. *Conjuntos.*

Se define como una estructura de datos que contiene una colección desordenada de distintos valores. Las operaciones que se pueden hacer son:

a) Prueba de pertenencia membresía. Significa investigar si un valor pertenece a un conjunto.
b) Inserción y supresión de valores. Significa insertar un valor en el conjunto, verificando que no se encuentre en el mismo y borrar un valor del conjunto verificando que se encuentre en el mismo respectivamente.
c) Unión de conjuntos. *Significa crear un conjunto* C3 *con todos los valores de un conjunto* C1 y del conjunto C2, eliminando los duplicados.
d) Inserción de conjuntos. Significa crear un conjunto C3 que contenga los valores que son miembros del conjunto C1 y C2.
e) Diferencia de conjuntos. Significa crear un conjunto C3 que contenga solo los valores que hay en C1 pero no en C2(operación de diferencia)

6. Archivo.

Existen diferentes tipos de archivos pero en este caso veremos el archivo secuencial. Este tipo de archivo se define como una estructura de datos que esta compuesta de una secuencia lineal de componentes del mismo tipo. Es de longitud variable, sin un límite máximo fijo, la forma de declarar un archivo secuencial varia de un lenguaje a otro. Las operaciones que podemos hacer con el archivo son:

a) Abrir. Está operación da el nombre de un archivo y su modo de acceso (lectura o escritura). Si el modo es de lectura entonces el archivo ya existe y esta listo para leerse, pero si lo abrimos de escritura requiere que el sistema operativo cree un nuevo archivo vacío listo para escribir sobre el, colocando un apuntador al inicio del mismo, pero si intenta abrir uno de escritura ya existente entonces borra todo lo que contenga y coloca el apuntador al inicio del mismo.
b) Leer. El objetivo es transferir los contenidos del componente actual a una variable designada por el programa.
c) Escribir. Crea un nuevo componente en la posición actual del archivo siempre al final y transfiere los contenidos de una variable designada por el programa a un nuevo componente.
d) Prueba de fin de archivo. Verifica que el apuntador del archivo este siempre al final para una escritura y si es para una lectura entonces verifica que el apuntador este siempre al inicio del archivo.
e) Cerrar. Implica notificar al sistema operativo que el archivo puede desprenderse del programa, algunos lenguajes interactúan con el sistema operativo permitiéndoles que no se cierre el archivo, sino que el propio sistema operativo se encargue de ello.

3.5. Implementación de tipos de estructuras de datos.

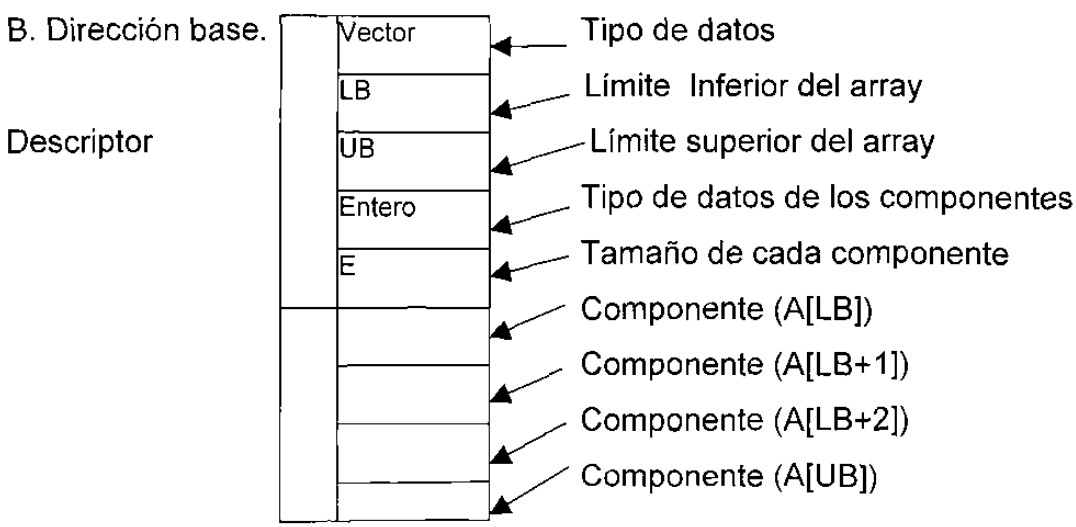
En esta sección tiene como objetivo mostrar como se representan en memoria los difererentes tipos antes expuestos dependiendo de la arquitectura de la máquina y del lenguaje. Existen 2 organizaciones para representar el tipo de memoria las cuales son:

1. Secuencial. Consiste en seleccionar un grupo de localidades contiguas en el almacenamiento formando un bloque único para manipular la estructura.
2. Encadenada. Consiste en proporcionar un conjunto de pequeños bloques de memoria enlazadas en cierta secuencia para formar la estructura de dato.

También es esta sección se analiza la representación de las diferentes operaciones que se pueden hacer con los tipos de datos. A continuación se exponen los diferentes tipos antes mencionados.

1. Arreglos o vectores.

La representación en memoria consiste en un bloque donde se representan en forma secuencial los componentes en forma de vector y algunos casos un descriptor (registro que mantiene cierta información importante de un tipo de datos). La organización de un vector es la siguiente:



Para representar la operación de selección de un componente realiza mediante la siguiente formula:

Loc(A[i])= B + D + (I - LB) * E

El objetivo de está es obtener la localidad de memoria donde inicia el almacenamiento del componente solicitado(A [i]). Para ello utiliza la dirección base (B) del bloque de memoria donde inicia el almacenamiento del descriptor y de los elementos del vector, a la misma se le suma el tamaño del descriptor (D) además se le agrega el producto resultante de la diferencia entre el componente i y el límite inferior del subíndice (LB) multiplicado por el tamaño del componente (E).

Las demás operaciones son realizadas directamente por el hardware o por el software, incluyendo las que se utilizan para el manejo de operaciones entre componentes. Cuando se trata de una asignación entre estructuras completa la implementación consiste en la copia de un bloque de memoria.

2. Registros.

La representación en memoria de un registro consiste en utilizar un bloque de localidades de memoria contiguas, generalmente este bloque solo almacena los componentes y no existe descriptor. La organización de un registro es la siguiente:

Dirección base (B)

Componente uno
Componete dos
Componente n

Para representar la operación de seleccionar se realiza mediante la siguiente forma:

Loc (A[i]) = B+S(tamaño del componente) desde j= 1 hasta i-1

El objetivo de está formula es obtener la localidad de memoria donde inicia el almacenamiento del componente solicitado (A [i]). Para ello utiliza la dirección base del bloque de memoria donde inicia el almacenamiento de los elementos del registro, a la misma se le suma el tamaño del primer componente(S_1) y luego el tamaño del segundo (S_2) y así sucesivamente hasta encontrar el componente deseando.

Las demás operaciones son realizadas directamente por el hardware o por el software, incluyendo las que se realizan para el manejo de operaciones entre componentes. Cuando se trata de una asignación entre estructuras completa la implementación consiste en la copia de un bloque de memoria.

3. Cadena de caracteres.

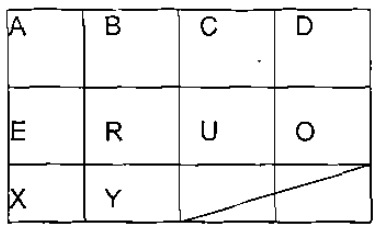
Para cada uno de los tres métodos para manejar cadena de caracteres utilizan representación de almacenamiento diferente, como se muestra a continuación:

a) Longitud declarada fija. La representación es utilizada para un vector empaquetado de caracteres.

A	B	C	D
E	R	U	O
X	Y		

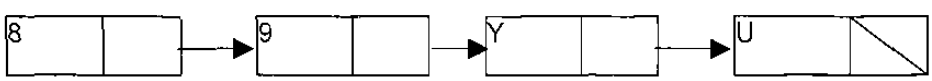
Código de caracteres de 8 bits empaquetados 4 por palabra y extendidos con espacios en blanco a la longitud fija de 12 caracteres.

b) Longitud variable a un límite declarado. La representación del almacenamiento utiliza un descriptor que contiene la longitud máxima declarada y la longitud actual de la cadena almacenada en el dato objeto.



Código de caracteres de 8 bits empaquetados 4 por palabra sin extensión.

c) Longitud ilimitada. Para esta se necesita una representación de una unidad de almacenamiento como un descriptor que contiene la longitud actual de la cadena.



Código de caracteres de 8 bits empaquetados 2 por palabra con apuntador a la siguiente palabra.

Para representar las operaciones sobre cadena generalmente son simuladas por el software del lenguaje.

4. Datos objeto y apuntadores dinámicos.

Un dato objeto indicador está representado como una localidad de almacenamiento que contiene la dirección base del bloque de almacenamiento que representa el dato objeto indicado por el indicador. Dos representaciones de almacenamiento principales se utilizan para valores de indicador.

a) Dirección absoluta. Un valor del indicador puede representarse como la dirección de memoria real del bloque de almacenamiento para el dato objeto.

b) Dirección relativa. Un valor del indicador puede representarse como una equivalencia de la dirección base de algún bloque de almacenamiento en un lote o área dentro de la cual el dato objeto se coloca.

5. Conjuntos.

Para representar los conjuntos se representa por una cadena de bits de longitud N, donde el bit i-ésimo en la cadena es uno sí esta dentro del conjunto y 0 si no esta. Para representar la operación de inserción de un elemento dentro de un conjunto consiste en la colocación de un 1 en el bit apropiado, la supresión consiste en la colocación del bit apropiado en 0.

Las operaciones de unión, intersección y diferencia sobre conjuntos puede representarse por medio de las operaciones booleanos sobre cadenas de bits: la operación booleana OR de las 2 cadenas de bits representa unión, AND representa intersección y el AND de la primera cadena y el complemento de la segunda representa la diferencia.

6. Archivos.

Para representar este tipo de datos en memoria, simplemente se almacenan sus componentes en forma secuencial. En la mayoría de los sistemas de computación el sistema operativo tiene la responsabilidad primaria para la implementación de archivos. Cuando un programa abre un archivo durante su ejecución debe proporcionar almacenamiento para una tabla de información del archivo y para un buffer. La operación de lectura del sistema operativo almacena información acerca de la localidad y las características del archivo en la tabla de información del archivo.

El buffer se usa como área de almacenamiento. Supongamos que el archivo esta abierto de modo escritura, entonces los datos que quieren escribirse en el archivo se colocan en el buffer, ninguna transferencia real de datos al archivo toma lugar hasta que se hayan ejecutado suficientes operaciones de escritura para permitir que un bloque completo de componentes se hayan acumulado en el buffer. En ese momento el bloque de componentes se transfiere del buffer del

dispositivo del almacenamiento externo. Sin embargo cuando se lee un archivo ocurre el proceso inverso los datos se transfieren del archivo al buffer en bloques de componentes. Cada operación de lectura ejecutada por el programa transfiere un solo componente del buffer a la variable del programa. Y el buffer se vuelve a llenar conforme se necesite.

3.6. Tipos de datos abstractos definidos por el programador.

Muchas veces un programador necesita elaborar programas grandes donde generalmente le resulta insuficientes los tipos de datos con los que cuenta el lenguaje y es necesario construir tipos de datos definidos por él. Nace el concepto de un subprograma que es precisamente una operación abstracta definida por el programador. La especificación abarca los siguientes conceptos.

-Nombre del subprograma.

-Números de argumentos, su orden, y el tipo de dato de cada uno.

-El número de resultados, su orden y el tipo de dato de cada uno.

-Acción realizada por el subprograma.

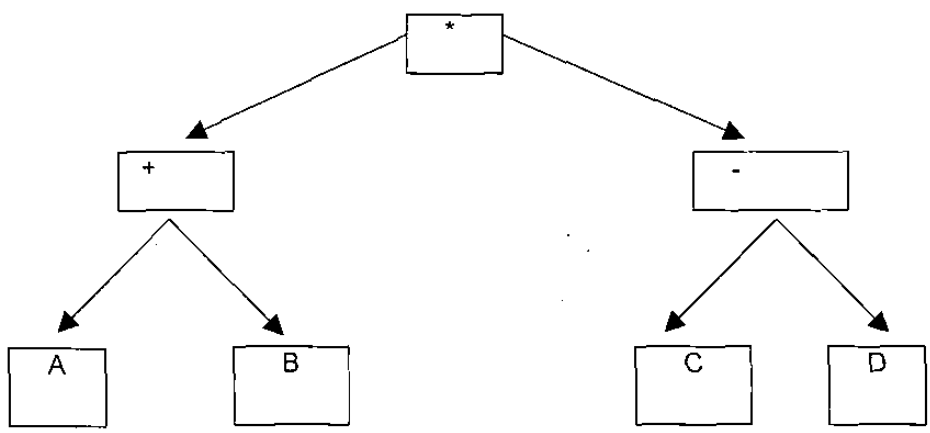
La implementación se define por el cuerpo del subprograma, el cual consiste de declaraciones, datos locales que definen las estructuras de datos usadas por el subprograma y de proposiciones que definen las acciones que se van a tomar cuando el subprograma se ejecuta.

4. CONTROL DE SECUENCIA

El objetivo de este capítulo es dar a conocer que dentro de un programa es necesario considerar las sentencias ya que describen el aspecto de un programa y se caracterizan en tres grupos que son: Expresiones, proposiciones y grupos de proposiciones y subprogramas.

4.1. Expresiones.

Una expresión representa la secuencia de las operaciones, aunque a veces esto trae consecuencias y es necesario explicar algunas notaciones importantes, utilizando el concepto de construcción de árboles. A continuación se muestra un árbol que servirá de ayuda para explicar y escribir las notaciones:



Árbol.

Notaciones:

a) Notación prefijo ordinaria.

En esta notación se escribe el símbolo de la operación primero, seguido de los operandos de izquierda a derecha. Pero si un operando es en sí mismo una operación con operandos, se aplican las mismas reglas. En está notación simplemente se encierra la secuencia de operandos en paréntesis, separado por comas.

Ejemplo: *(+A,B),-(C,A))

b) Notación prefijo Polaca de Cambrige. En está notación el paréntesis izquierdo que sigue a un símbolo de operación se mueve para precederlo inmediatamente y se suprimen las comas que separan los operandos.

Ejemplo: (*(+AB)(-C,A))

c) Polaca sin paréntesis. En está notación es igual que la anterior excepto que los paréntesis se omiten

Ejemplo:* + A B –C A

d) Notación postfijo. Es similar a la notación prefijo excepto que el símbolo de operación sigue a la lista de operandos.

Ejemplo: ((A, B)+,(C,A)-)* ó AB+CA-*

e) Notación infijo. Está notación es adecuada solamente para operaciones binarias.

Ejemplo: (A+B)*(C-A)

De acuerdo a la última notación origina una ambigüedad, por ejemplo la siguiente expresión: A*B+C, por ello se recurre a las reglas de precedencia las cuales nos indican que operadores en una expresión ocurre primero y que operandos ocurren después. Sin embargo hay operadores que tienen el mismo orden de precedencia y en ese momento que nace el concepto de asociatividad, el cual nos dice que las operaciones se deben de ejecutar de izquierda a derecha o viceversa cuando 2 o más operadores tiene el mismo grado de precedencia en una expresión.

4.2. Instrucciones o proposiciones.

El objetivo es entender los mecanismos básicos con los que se controla la secuencia en la cual se ejecutan las proposiciones individuales dentro de un programa.

Hay diferentes formas de proposiciones que a continuación se describen cada una de ellas, utilizando el lenguaje C para los ejemplos tenemos las siguientes:

a) Proposiciones condicionales. Está proposición expresa la alternación de dos o más proposiciones o la ejecución opcional de una sola proposición, la cual se determina por medio de una prueba sobre alguna condición, algunas formas diferentes son las siguientes:

1. Ramificación Simple. **IF (expresión1) entonces expresión2**

Donde la expresión1 es evaluada para determinar si es cierta entonces se realiza la expresión 2 se caso contrario no pasa nada.

Ejemplo**: if (cuenta= 0)**

saldo = 0

Esto significa que si cuenta es igual a 0 entonces saldo es igual a 0 de caso contrario no pasa nada.

2. Ramificación doble. **Expresion1?expresión2:expresión3**

Donde la expresión1 es evaluada para determinar si es cierta o falsa si es verdadera entonces se realiza la expresion2 de caso contrario se ejecuta la expresion3.

Ejemplo: **saldo= (cuenta= 0) ¿ 0: totsa/cuenta;**

Esto significa que si cuenta es igual a 0 entonces saldo tendrá un valor de 0 de caso contrario saldo tendrá un valor del resultado de totsal/cuenta.

3. Proposición case. **Switch(expresión) sentencia**

Está proposición sirve para escoger un grupo de proposiciones entre varios grupos disponibles. La selección se basa en el valor de una expresión que se incluye en la sentencia switch

Ejemplo: **Switch(color)**

```
{ case 'r':printf("rojo");
  case 'b':printf("blanco");
 defaul:printf("Error");
 }
```

Esto significa que está instrucción evalúa el valor de color si es r entonces despliega rojo, si en b despliega blanco, pero si no es alguno de ellos despliega "Error".

a) Proposiciones de iteración. El objetivo de estas proposiciones es ejecutar repetitivamente una o más proposiciones hasta que se cumpla una condición.

Algunas formas diferentes son las siguientes:

1. Proposición While.

```
While (expresión1)
{ expresión2
  expresión3
  expresión4
}
```

Esto significa que mientras la expresión1 sea verdadera entonces se ejecutan la expresión2, expresión3 y la expresión4.

Ejemplo:

```
Dig= 0;
While(dig<=9)
{ printf("%d",dig);
dig++;
}
```

Esto significa que se van a estar imprimiendo el valor de dig desde 0 hasta 9.

2. Proposición for. **For(expresión1; expresión2; expresión3) sentencia**

En donde expresión1 es utilizada para inicializar algún parámetro que controla la repetición del ciclo, expresión2 representa una condición que debe ser satisfecha para que continúe la ejecución del ciclo y la expresión3 se utiliza para modificar el valor del parámetro inicialmente asignado por expresión1.

Ejemplo:

```
For(dig=0; dig<=9; dig++)
  Printf("%d", dig);
```

Esto significa que se van a estar imprimiendo el valor de dig hasta que el mismo tenga un valor de 9.

4.3. Subprograma simple (call-Return).

En esta sección el objetivo es entender el funcionamiento de un subprograma en un lenguaje de programación, así como también la forma en que se implementa en memoria. Para entenderlo es necesario dividirlo en dos partes las cuales a continuación se explican:

1. Especificación.

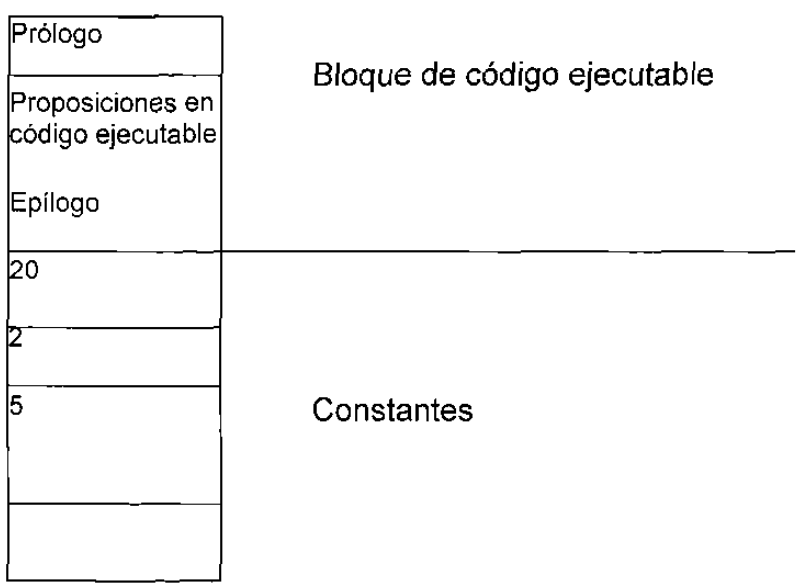
Un subprograma está compuesto de un solo programa principal el cual durante la ejecución puede llamar a varios subprogramas, los cuales a su vez pueden cada uno llamar a otro subprograma y así sucesivamente a cualquier profundidad. Cada subprograma en cualquier punto se espera a que termina su ejecución y que regrese el control al programa llamado se interrumpe en forma temporal. Cuando la ejecución de un subprograma se completa, la ejecución del programa llamado se resuma en el punto que sigue de inmediato al llamado del subprograma.

2. Implementación.

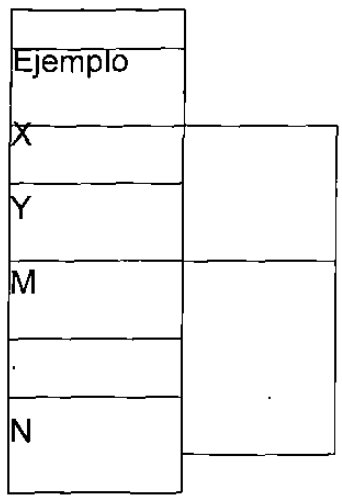
Para entender la implementación de un subprograma necesitamos conocer las partes por las que esta formada. Suponga la siguiente función en el lenguaje de programación C.

```
Int ejemplo(int x, int y)
{ const max= 20;
 int m[max];
float n;
n=max;
x= 2*x+m[5];
}
```

La implementación de esta función se implementa en dos partes: en el segmento de código y el registro de activación que es el siguiente:



Segmento de código.



Registro de activación.

El segmento de código es una área de almacenamiento invariable durante la ejecución y se crea por medio de un traductor, que mantiene la información en forma estática y contiene: código ejecutable, constantes, epílogo (su función es regresar resultados y liberar almacenamiento del registro de activación) y prologo(su función es transmitir los resultados)

El registro de activación es una área de almacenamiento que se crea de nuevo cada vez que el subprograma se llame y se destruya cuando el programa regresa, contiene datos locales y parámetros.

Además es necesario dar a conocer los siguientes conceptos:

a) AIA (Apuntador a la instrucción actual). Es un apuntador que hace referencia a la instrucción que se va a ejecutar. El interprete actúa trayendo la instrucción designada por el AIA, actualiza AIA para que apunte a la siguiente instrucción de la secuencia.
b) AAA (Apuntador al ambiente actual). Es una apuntador que hace referencia al registro actual que se mantiene durante la ejecución de un dato objeto.

Una vez explicado lo anterior se procede a explicar como se implementa un subprograma Call-Return en memoria.

Primero se crea un registro de activación para el subprograma principal A el **AAA** se le asigna un apuntador. A el **AIA** se le asigna un apuntador para la primera instrucción en el segmento de código para el programa principal. El interprete va a trabajar, trayendo y ejecutando las instrucciones como lo designo el **AIA** .

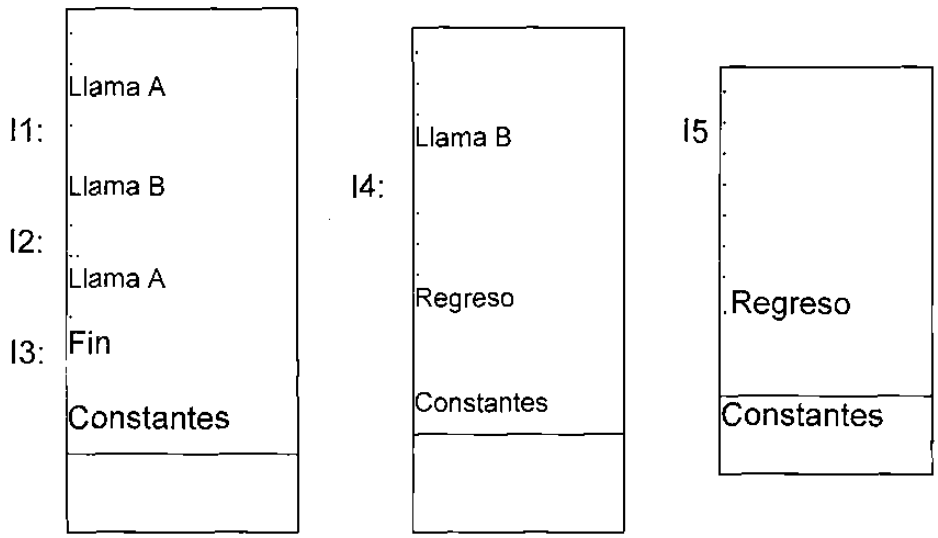
Cuando se alcanza una instrucción **CALL** de un subprograma se crea un registro de activación para el subprograma y se asigna una apuntador para el **AAA**. A el **AIA** se le asigna un apuntador para la primera instrucción del segmento de código para el subprograma y el interprete continua desde ese punto, ejecutando instrucciones en el programa.

Si el subprograma llama a otro subprograma se hacen nuevas asignaciones para establecer el **AIA y AAA** para la activación de ese subprograma. Para poder regresar correctamente de una llamada al subprograma, los valores del **AIA y AAA** deben de salvarse en el punto de retorno (formado por dos valores: **ai y aa**) que se incluye en el registro de activación, antes de que se asignen nuevos valores a **AIA y AAA.**

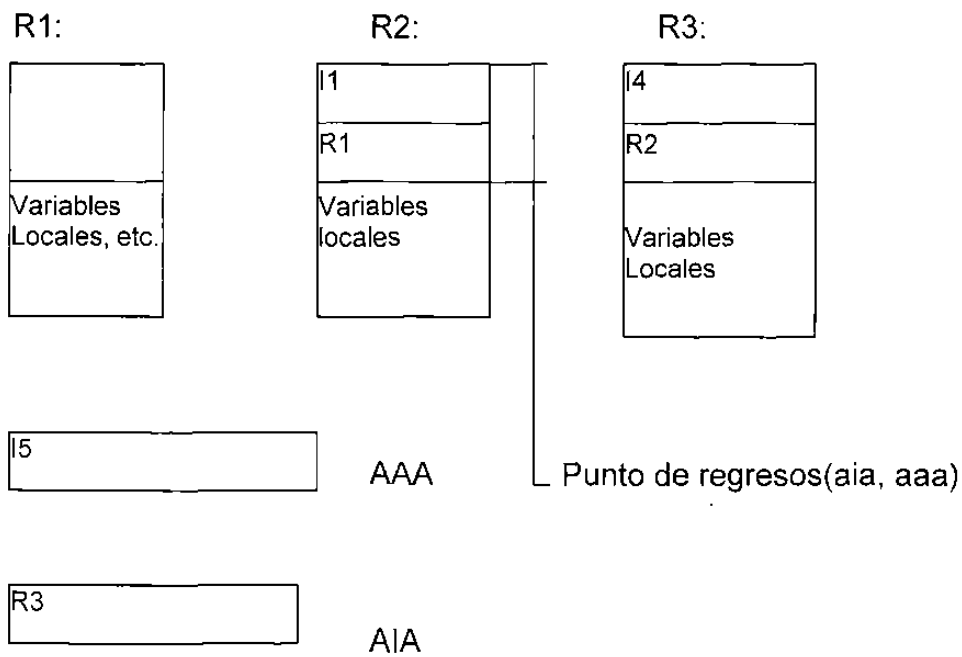
Cuando se alcance una instrucción **RETURN** que termine una activación de un subprograma los valores anteriores del **AIA y del AAA** que se salvarón cuando el subprograma se llamó deben repararse y reinstalarse.

Después que la instrucción **CALL** crea un registro de activación almacena los valores anteriores **(ai,aa),** del **AIA** y del **AAA** en el punto de regreso y asigna el nuevo **(ai,aa)** a el **AIA** y **AAA**; Así se efectúa la transferencia del control al programa llamado.

A continuación se muestra la siguiente figura.



Segmento de código almacenado en forma estática.



Registro de activación.

4.4. Subprograma Recursivo.

En esta sección el objetivo es entender el funcionamiento de un subprograma recursivo en un lenguaje de programación, así como también la forma en que se implementa en memoria.

Para entenderlo es necesario dividirlo en dos partes las cuales a continuación se explican.

1) Especificación.

La recursión es el mecanismo de control primario para repetir las secuencias de proposiciones, remplazando la iteración de la mayoría de los lenguajes. La única diferencia entre una llamada recursiva y un normal es que la primera crea una segunda activación del subprograma durante el tiempo de vida de la primera activación, si la segunda activación conduce a otra llamada recursiva entonces pueden existir 3 activaciones al mismo tiempo y así sucesivamente.

2) Implementación.

Para el manejo de almacenamiento es directo, utiliza una pila central, el último elemento creado en la pila debe ser el primero que se borre. La implementación prosigue de la siguiente forma: al principio de la ejecución del programa se reserva un gran bloque de almacenamiento para la pila central, el registro de activación del programa principal se convierte en el fondo de la pila.

Cuando se llama a un **subprograma A,** se coloca almacenamiento para este registro de activación, cerca de aquel registro de activación del programa principal. Si **A** llama a **B**, el registro de activación de **B** se coloca adyacente al de **A**, si **B** llama a **C**, el registro de activación de **C** se coloca adyacente al de **B** y así sucesivamente.

Cuando **C** termina y regresa el control a **B**, cualquier almacenamiento colocado mas allá del de **C** en la pila central debe haberse liberado.

El almacenamiento de **C** se libera y después el de **B** y así sucesivamente. Y cada registro de activación contiene un punto de regreso para almacenar los valores de **AIA y AAA**.

4.5. Excepciones.

Durante la ejecución de un programa en vez de continuar con la ejecución normal del programa, necesita llamarse a un subprograma para ejecutar un proceso especial llamado manejador de excepciones.

La acción de notar la excepción, interrumpir la ejecución del programa y transferir el control al manejador de excepciones se llama surgir una excepción.

Después el manejador completa el proceso de una excepción y está listo para terminar puede tomar algunas de las siguientes alternativas:

1. ¿Debe regresar el control al punto donde se elevó la excepción?
2. ¿Debe terminar el subprograma que contiene el manejador en sí, pero terminado en forma normal para que parezca como si nada hubiera pasado?

Cuando ocurre una excepción algunos lenguajes de programación soportan excepciones mediante subrutinas implementadas por el mismo o los mismos programadores las definen.

4.6. Corrutinas.

En está sección el objetivo es entender el funcionamiento de una corrutina en un lenguaje de programación, así como también la forma en que se implementa en memoria. Para entenderlo es necesario dividirlo en dos partes las cuales a continuación se explican:

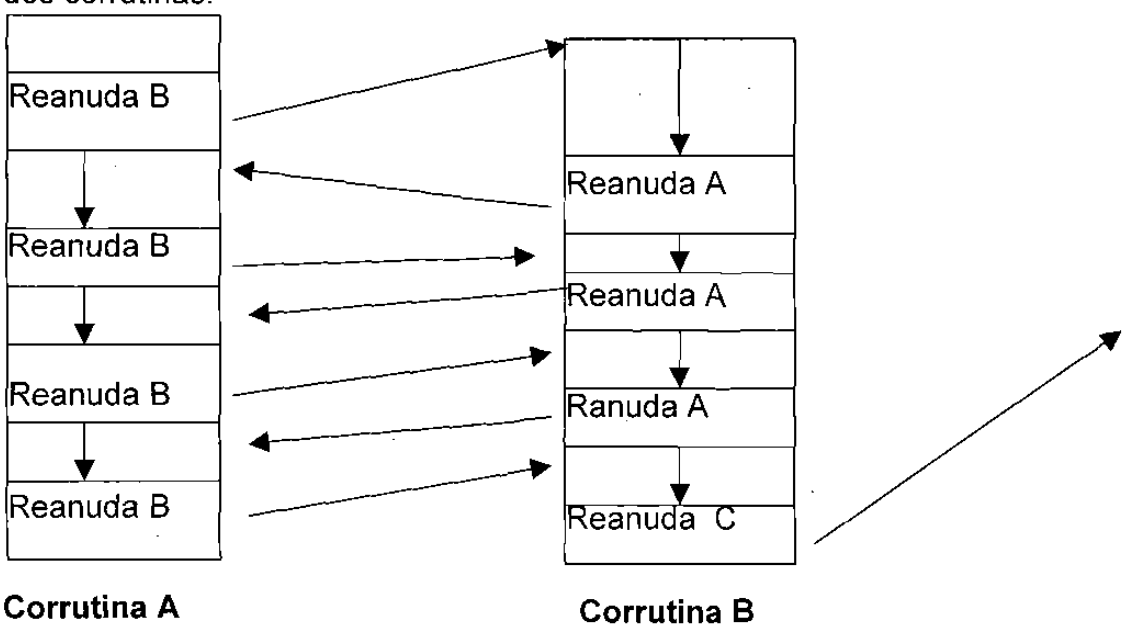
1) Especificación.

Una corrutina es un subprograma que no se ejecuta completamente antes de mandar el control al programa que lo llamo. Cuando una corrutina recibe el control de otro subprograma se ejecuta en forma parcial. La ejecución de la corrutina se suspende cuando regresa el control y en un punto posterior el programa que lo llamo puede reanudar la ejecución de la corrutina en el punto en el cual se suspendió la ejecución.

El nombre de la corrutina se deriva de su simetría es decir en vez de un programa que llama y un programa llamado los dos programas parecen iguales(dos subprogramas intercambian el control de aquí para allá conforme cada uno se ejecuta) sin que ninguno de los dos controle al otro en forma clara.

Generalmente las corrutinas se activan con la proposición **resume** que transfiere el control entre corrutinas, especifica la reanudación de alguna activación particular de la corrutina.

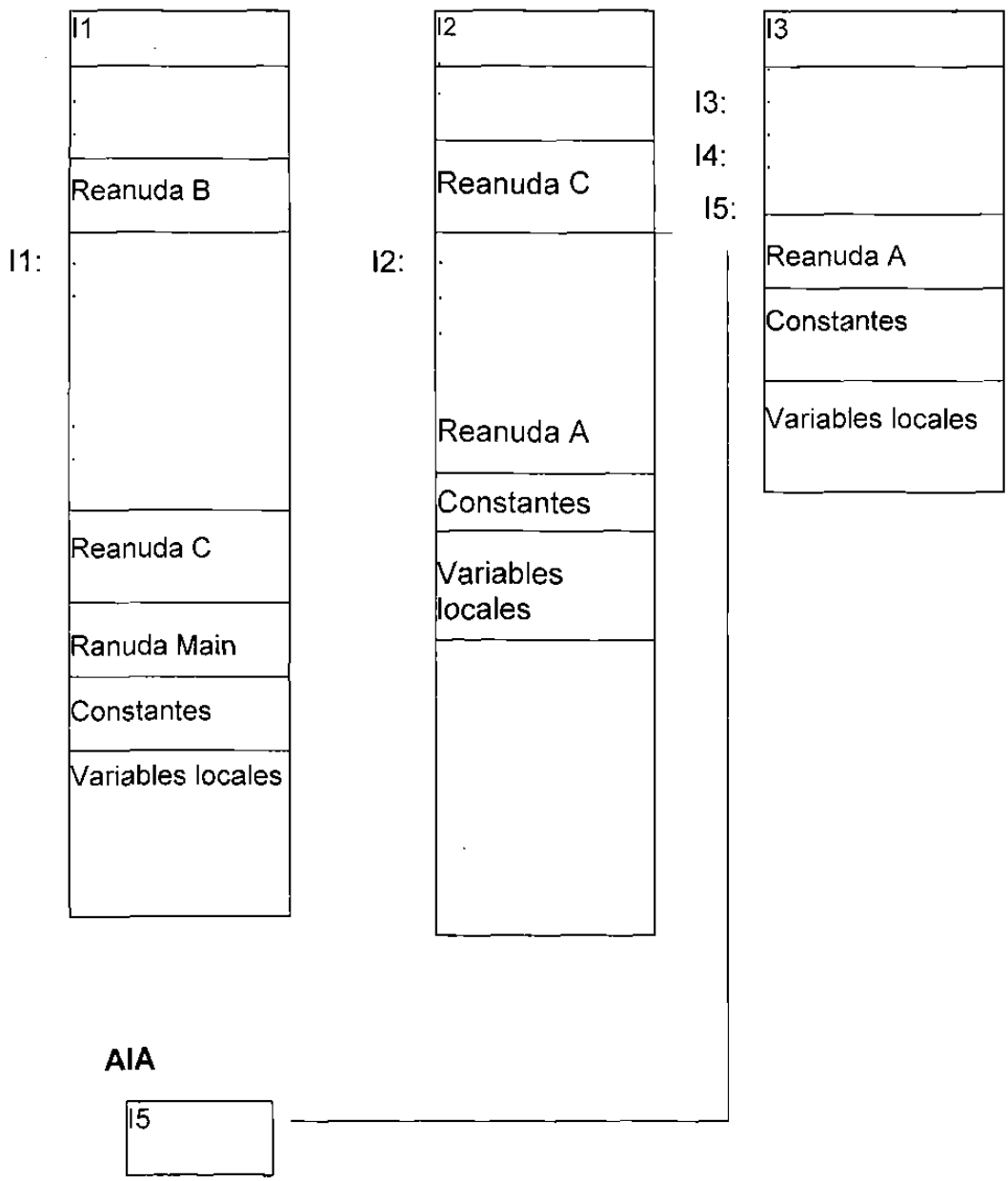
A continuación la siguiente figura ilustra la transferencia de control entre dos corrutinas.



2) Implementación.

Un registro de activación es colocado en el almacenamiento en forma estática al principio de la ejecución, como una extensión del segmento de código para la corrutina. Una sola localidad ahora punto de reanudación se reserva en el registro de activación para salvar el estado anterior **a1** de **AlA** cuando una proposición resume transfiere el control para otra corrutina. Sin embargo de acuerdo a la siguiente figura como el punto de regreso en el **subprograma B** se usa para almacenar el valor **ai** por **B** en sí mismo. La ejecución de una instrucción **resume B** en la **corrutina A** incluye dos pasos:

1. El valor de **AIA** se salva en la localidad del punto de reanudación del registro de activación para **A**.
2. El valor de **ai** en la localidad del punto de reanudación de **B** se trae del registro de activación de **B** y se asigna al **AIA**, para efectuar la transferencia de control a la instrucción adecuada en **B**.



4.7. Tareas y concurrencia.

Tarea la podemos definir como un subprograma que se puede ejecutar en forma concurrente con otro. La idea básica de las tareas es muy simple, imaginemos un **subprograma A** que se esta corriendo en forma normal. Si **A** llama al **subprograma B**, la ejecución de **A** se suspende mientras **B** se ejecuta. Sin embargo, si **B** se inicia como tarea, la ejecución de **A** continúa mientras **B** se ejecuta.

La secuencia original de ejecución se divide en dos secuencias de ejecución paralelas, ya sea **A** o **B** ambas podrían iniciar tareas posteriores permitiendo que coexista cualquier número de secuencias de ejecución paralelas, en general cada tarea se considera un dependiente de la tarea que la inicio. Cuando una tarea está lista para terminar debe esperar hasta que todas sus dependientes hayan finalizado antes de que pueda terminar. Así la división en secuencias de ejecución múltiples se invierte conforme las tareas terminan, volviéndose menos y menos secuencias hasta que finalmente solo queda una secuencia.

La sintaxis y semántica de las tareas varían en cada lenguaje que las proporcionen.

5. CONTROL DE DATOS

El objetivo es dar a conocer cuales son los elementos importantes en los lenguajes de programación para controlar el flujo de los datos de un programa.

5.1 Nombres y ambientes referenciales.

Solo hay dos formas en que un dato objeto se puede hacer disponible como un operando para una operación:

1. Transmisión directa. Un dato objeto calculado en cualquier punto como el resultado de una operación puede transmitirse en forma directa a otras operaciones como un operando.
2. Referencia a través de un dato objeto nombrado. A un dato objeto puede dársele un nombre cuando se crea y el nombre puede usarse entonces para designarlo como operando de una operación.

La referencia a través de un objeto nombrado es de nuestro interés porque incluye el uso de nombres y la referencia de nombres por consiguiente nos elaboramos la siguiente pregunta: ¿Qué tipos de nombres se ven en los programas?.

Cada lenguaje difiere, pero algunas de las siguientes lo soportan diversos lenguajes son: nombres para subprogramas, nombres de variables, nombres de parámetros formales, nombres para constantes definidas, nombre para las excepciones y nombres para las operaciones primarias como SQRT, SEN, *,+, entre otras.

Un nombre en cualquiera de las categorías anteriores puede llamarse un nombre simple y un nombre compuesto es un nombre para un componente de una estructura de datos, escrito como un nombre simple que designa a la estructura de datos completa, seguido de una secuencia de una o más operaciones de selección que seleccionan el componente particular de la estructura nombrada.

El control de datos se relaciona en gran parte con la unión de identificadores es decir simples a datos objeto y subprogramas tal unión se le llama asociación y puede representarse como un par que consta de identificador y de su dato objeto asociado o subprograma.

Con el uso de creación y destrucción de las asociaciones que ocurren en el transcurso de un programa vemos los siguientes conceptos principales de control de datos:

1. Ambientes referenciales. Es el conjunto de asociaciones e identificadores que tienen un programa o subprograma disponibles apara usarse en las referencias durante su ejecución. Se determina cuando se crea la activación del subprograma y continua sin cambio durante el tiempo de vida de la activación el ambiente referencial de un subprograma puede tener varios componentes:

a) Ambiente referencial local. Es el conjunto de asociaciones creadas sobre la entrada a un subprograma que representa los parámetros formales, las variables locales y los subprogramas definidos solo dentro de ese subprograma.

b) Ambiente referencial foráneo. Es el conjunto de asociaciones para los identificadores que pueden usarse dentro de un subprograma pero que no se crean sobre la entrada de el.

c) Ambiente referencial global. Es el conjunto de asociaciones para los identificadores que pueden usarse dentro de un subprograma pero que se crean al inicio de la ejecución del programa principal.

d) Ambiente referencial predefinido. Es el conjunto de asociaciones que pueden usarse en un programa pero que no se crean en ningún lado del programa por parte del programador sino que están predefinidas por el mismo lenguaje.

2. Visibilidad. Una asociación para un identificador se dice que es visible dentro de un subprograma si es parte del ambiente referencial de ese subprograma.

Una asociación que exista pero que no sea parte del ambiente referencial de ese subprograma que está en ejecución actualmente se dice que esta oculto de ese subprograma.

5.2. Expansión dinámica y estática.

La expansión dinámica de una asociación para un identificador es ese conjunto de activaciones de subprogramas en las cuales la asociación esta visible durante la ejecución. La regla de expansión dinámica define la expansión dinámica de cada asociación en términos de curso dinámico de la ejecución del programa. Por ejemplo, una regla de expansión dinámica establece que la expansión de una asociación creada durante la activación del **subprograma P** incluye no solo esa activación sino cualquier activación de un subprograma llamado por **P**, o llamado por un subprograma llamado por **P** y así sucesivamente a menos que activaciones de subprogramas posteriores definan una nueva asociación local para el identificador que oculta la asociación original. Sin embargo si el lenguaje soporta que el programador realice declaraciones de identificadores entonces cada una de ellas tiene expansión estática. Y las reglas de expansión estática relatan referencias con declaraciones de nombres en el texto del programa.

5.3. Estructura en bloques.

Podemos definir un bloque como un subprograma o el programa principal. En un programa estructurado en bloques, cada subprograma o programa se organiza en un conjunto de bloques anidados. La anidación de bloques se completa permitiendo que la definición de un bloque contenga completamente las definiciones de otros. En el nivel mas exterior, un programa

consiste en un bloque *sencillo* que define el programa principal. Dentro de este bloque están otros que definen subprogramas que pueden llamarse del programa principal en su interior podría haber otros bloques que definieran los subprogramas que puedan llamarse de los subprogramas del primer nivel y así sucesivamente. Sin embargo es necesario establecer reglas de expansión estática asociadas un programa estructurado en bloques las cuales son las siguientes:

1. Las declaraciones al inicio de cada bloque definen el ambiente referencial local del bloque. Cualquier referencia a un identificador dentro del cuerpo del bloque (sin incluir ningún bloque anidado) se considera una referencia a la declaración local para el identificador para el identificador si es que existe.

2. *Si un identificador se referencia dentro del cuerpo del bloque y no existe* ninguna declaración local, entonces la referencia se considera una referencia para una declaración dentro del bloque que encierra inmediatamente al primer bloque. Si no existe ninguna declaración allí entonces se refiere a una declaración en el bloque que encuentra inmediatamente a ese bloque y así sucesivamente. Si el bloque mas exterior se alcanza antes de que se encuentre una declaración, la referencia es a la declaración dentro del bloque exterior, finalmente si no se encuentra ninguna declaración ahí la declaración en el ambiente predefinido se usa, si hay, si no se toma como un error la referencia.

3. Si un bloque contiene otra definición de bloques cualquier declaración local dentro del bloque interior o cualquier bloque que contenga están completamente ocultos del bloque exterior y no puedan referenciarse de el.

4. Un nombre del bloque se convierte en la parte del ambiente referencial local del bloque que contiene.

5.4. Datos compartidos.

Un dato objeto que es local se usa en operaciones solo dentro de un simple ambiente referencial local esto es, dentro de un simple subprograma muchos datos objeto deben compartirse entre varios subprogramas para que las operaciones en cada uno de los subprogramas puedan usar los datos.

Un dato objeto puede compartirse a través de diferentes formas las cuales son las siguientes:

1. Ambiente común. Esta compuesto por un conjunto de datos objeto (variables, constantes y tipos pero no subprogramas ni parámetros formales) que van a compartirse entre varios subprogramas es colocado en el almacenamiento como un bloque separado. Si dentro de un subprograma se requiere de este bloque se utiliza cierta sintaxis que varia de un lenguaje a otro para poder hacer referencia con algún componente del bloque.

2. Expansión dinámica. La establece considerando la cadena dinámica de activaciones es decir, considere el curso de la ejecución del programa: suponga las llamadas del programa principal al **subprograma A,** el cual llama a **B** que a su vez llama a **P**. Si en **P** referencia **X** y no existe ninguna asociación a **X** en **P**, entonces es natural volver al **subprograma B** que llamo a **P** y preguntar si **B** tiene una asociación para **X** . Si **B** la tiene entonces la asignación para **X** se usa, de otra manera, vamos al **subprograma A** que llamo a **B** y verificamos si **A** tiene una asociación para **X** si existe se usa de caso contrario se busca en el programa principal y se realiza lo mismo.

3. Expansión estática. La establece considerando las reglas de expansión estática aplicada a estructura en bloques antes mencionada.

4. Parámetros. Antes de explicar la transición mediante parámetros es necesario dar a conocer los siguientes conceptos:

a) Parámetro formal. Es una clase particular de datos objeto locales dentro de un subprograma. La definición del subprograma lista por lo regular los nombres y las declaraciones para los parámetros formales, considerando como ambiente local de ese subprograma.

b) Parámetro real. Es un dato objeto que se comparte con el subprograma transmitiéndose en forma explicita desde el que lo llama. Un parámetro real puede ser un dato objeto local que pertenece al llamador, puede ser un parámetro formal del que llama, puede ser un dato objeto foráneo visible para el que llama o el resultado que se desplegó por una función invocada por el que llama y transmitido de inmediato al subprograma llamado.

La transmisión de datos por medio de parámetros se puede hacer de diferentes formas:

a) Transmisión por referencia. Aquí el parámetro formal es un dato objeto local de tipo apuntador. El apuntador para el parámetro real se copia en esta localización del parámetro formal en el momento de llamar a un subprograma. Cuando el subprograma termina el apuntador del parámetro formal se suprime pero cualquier cambio de valor del parámetro real continua.

b) Transmisión por valor de resultado. Aquí el parámetro real y formal son del mismo tipo. El valor contenido del dato objeto del parámetro real se copia en el dato objeto del parámetro formal en el momento de llamar al subprograma cuando el subprograma termina los contenidos finales del dato objeto del parámetro formal se copian en le dato objeto del parámetro real, como si se ejecutara asignación explícita del parámetro real al formal.

c) Transmisión por valor. Aquí el parámetro real y formal se tratan de la misma forma que el anterior excepto que al finalizar de ejecutar el subprograma, el valor final del parámetro formal no se asigna al parámetro real.

d) Transmisión por valores constantes. Aquí el parámetro real y formal se tratan de la misma forma que el anterior excepto que no se permite ningún cambio en el valor del parámetro formal durante la ejecución del subprograma por consiguiente al finalizar de ejecutar el subprograma, el valor final del parámetro formal no se asigna al parámetro real.
e) Transmisión por resultado. Aquí el parámetro formal es una variable local sin valor inicial, cuando el subprograma termina el valor final del parámetro formal se asigna como el nuevo valor al parámetro real, como con la transmisión por el valor de resultado.

5.5. Tareas y datos compartidos.

Anteriormente en el capitulo 4 se hablo del concepto de tarea. En este capitulo se hace referencia a que como se comparten datos . No resulta fácil compartirse datos ya que pueden dar resultados no esperados, debido a que no existe una buena sincronización entre ambos pero para que no ocurra esto, existen algoritmos de exclusión mutua como son semáforos y monitores. Estos conceptos ya se han repasado en otra asignatura por lo cual no es necesario analizar cada uno de los algoritmos en forma detallada.

6. ADMINISTRACION DE LA MEMORIA

El objetivo de este capítulo es ver las diferentes implementaciones que manejan diferentes lenguajes para administrar la memoria.

6.1. Administración de la memoria en forma estática.

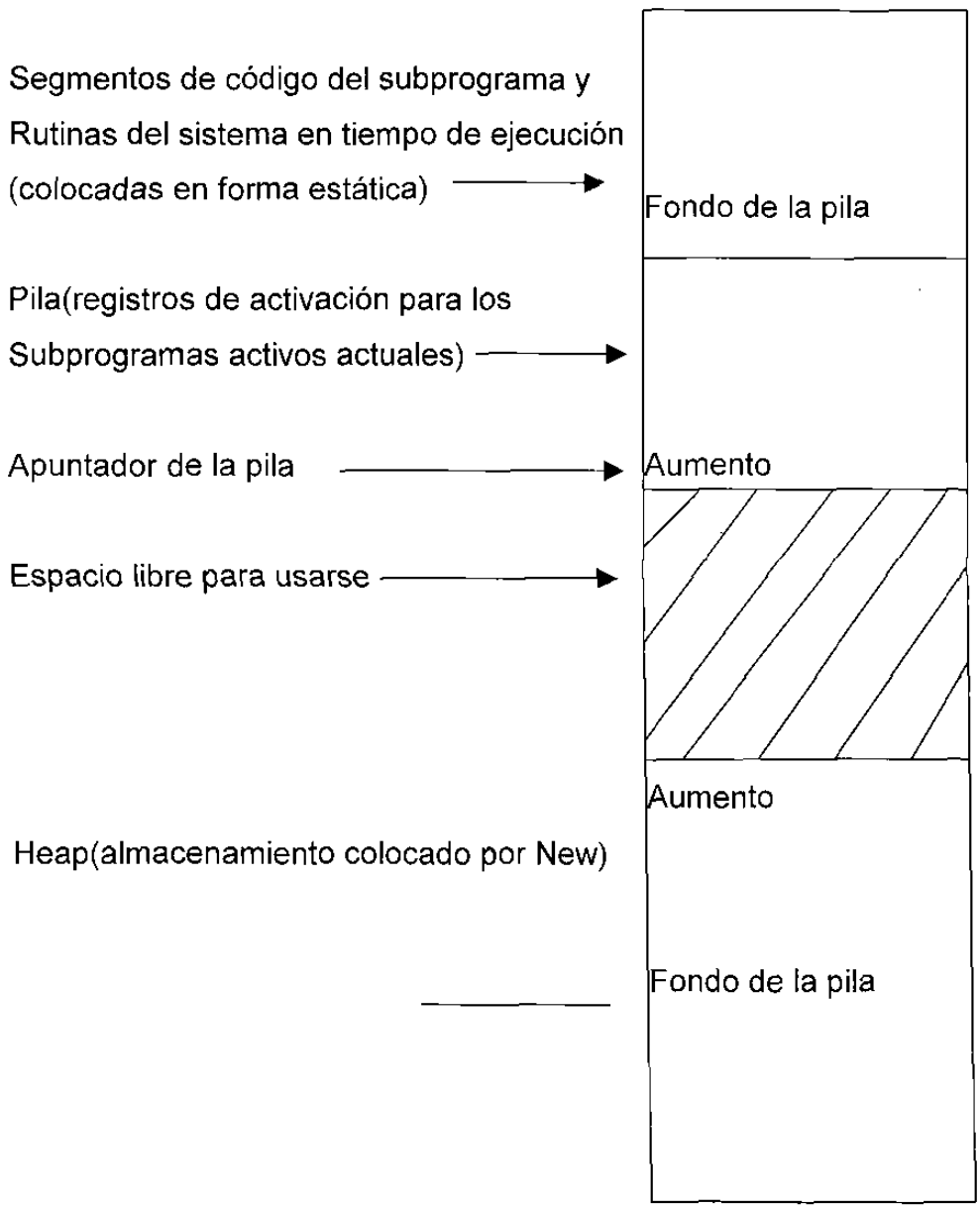
Se realiza durante la traducción y continua fija a través de la ejecución. Aquí cada subprograma se compila por separado y el compilador determina por separado el segmento de código incluyendo el registro de activación que contiene el programa compilado, sus áreas de datos, datos temporales y localidades en el tiempo de ejecución. La función del cargador coloca espacio en memoria para estos bloques compilados en el momento de cargar.

4.1. Administración basada en pilas.

Se realiza durante la ejecución. Utiliza una pila inicialmente la pila está libre. El almacenamiento libre al inicio de la ejecución se determina como un bloque secuencial en la memoria, conforme el almacenamiento se coloca se toma de las localidades secuenciales empezando al final de esta pila. El almacenamiento debe liberarse en orden contrario al de la colocación para que el bloque de almacenamiento que se esta liberando se encuentre siempre en la parte superior de la pila.

Es necesario un apuntador de pilas para que apunte a la siguiente palabra disponible de almacenamiento libre en el bloque de pila, que representa el que esta arriba actualmente en al pila, todo el almacenamiento en uso descansa en la pila abajo de la localidad apuntada por el apuntador. Cuando un bloque de K localidades va a colocarse se mueve al apuntador K localidades mas allá de área de la pila. Cuando se libera un bloque de K localidades, el apuntador se mueve de regreso K localidades. Cuando se llama un subprograma se crea un nuevo registro de activación sobre la parte superior de la pila. La terminación de una activación causa se libera un gran espacio de almacenamiento en la pila.

A continuación se muestra en la siguiente figura.



CONCLUSION

De acuerdo al trabajo realizado pretendo lograr una gama de conocimientos del alumno, obteniendo consigo una amplitud de conceptos básicos acordes al tema desarrollado.

Además facilita al docente los conceptos para que tenga la opción de añadir información a la misma y así elevar el nivel académico.

Es por ello que dentro de mi trabajo abordo temas necesarios para una mejor comprensión, resultando sencillo y factibles en el aprendizaje. Con ello se *da un ambiente de confianza, comunicación tanto para el maestro como para el* alumno.

Facilita así lo teórico de esta antología para las personas que *quieran* conocer y programar lenguajes de programación. Con ello se hace una constatación de la teoría con la práctica ya que una depende de la otra y resultan esenciales en el desarrollo de una actividad dada.

La antología que maneje resulta como un apoyo para el alumno como fuente de consulta propiciando con ello en el alumno una actitud investigadora enriqueciendo sus propios conocimientos día con día.

Con ello obtendremos resultados favorables para elevar el nivel académico proporcionando la información necesaria para que el alumno posea un panorama específico del tema señalado.

GLOSARIO

Ambigüedad. Permite una o mas interpretaciones diferentes de una construcción de un programa.

Bit. Representación de un estado ya sea uno(encendido alto voltaje) o un cero (apagado o bajo voltaje)

Buffer. Área de almacenamiento temporal.

Cliente / servidor. Significa que los servidores de archivos almacenan programas y datos que pueden ser compartidos por las computadoras cliente, distribuidas a todo lo largo de la red.

Computadora. Dispositivo electrónico capaz de recibir, almacenar, procesar y desplegar información.

Concurrente. Relativo a los procesos o programas que tiene lugar en un intervalo común de tiempo durante el cual pueda tener que compartir recursos alternativamente.

Ensamblador. Conjunto de instrucciones que convierten los programas de lenguajes ensamblador a lenguaje maquina.

Hardware. Es la herramienta de la Computadora es decir, los dispositivos como el teclado, pantalla, discos, memorias y las unidades de procesamiento.

Lenguaje de programación. Es cualquier notación para la descripción de algoritmos y estructuras de datos.

Lenguaje maquinal. Es el lenguaje natural de una computadora que consiste en una cadena de ceros y unos que instruyen a la computadora para que *ejecuten sus operaciones mas elementales una a la vez.*

Memoria. Lugar capaz de recibir información.

Multiprogramación. Un modo de operación que permite la ejecución intercalada de dos o más programas en un solo procesador.

Palabra. Conjunto de bits.

Pila. Es una lista que construye y se mantiene dé forma que el siguiente elemento de datos a recuperar sea el almacenado hace menos tiempo. Este método está caracterizado por un último en entrar primero en salir.

Sistema distribuido. *Se refiere a que la carga de trabajo de una organización* se distribuye sobre la red a los lugares en donde en realidad se ejecuta el trabajo de la organización es decir no es ejecutada en alguna instalación central de computo.

Sistema operativo. Conjunto de programas que controla la ejecución de diversos programas y ofrece servicios tales: asignación de recursos, planificación de gestión de datos y control de entrada-salida.

Software. Programa de computación que requiere la computadora.

Tabla de símbolos. Área de memoria donde se guarda información sobre todas las variables, identificadores y constantes que aparecen en el programa fuente.

Tiempo compartido. El uso concurrente de un dispositivo por una serie de usuarios.

Tokens. Conjunto de caracteres que forman un elemento de un lenguaje de programación como: un identificador, variables, operadores, caracteres especiales.

BIBLIOGRAFIA

Hm/.Deitel /P.J. Deitel. Como programar en C/C++. Segunda Edición Prentice Hall Hispanoamericana S. A.. 1995.

Alberto Rojas Ponce. Ensamblador básico. Computec Editores S. A. de C.V. 1993

William Stallings. Sistemas Operativos. Segunda Edición Prentice-Hall Madrid 1997.

Terrence W. Pratt. Lenguajes de Programación Diseño e Implementación. Tercera Edición. Prentice-Hall Hipanoamericana S.A. 1990

Robert W Sebesta. Concepts of Programming Languages. Segunda Edición The Benjamin/Commings Publishing Company Inc. 1993

RESUMEN AUTOBIOGRAFICO

Nombre del autor:	Ing. Adriana Ramírez Hernández
Grado a obtener:	Maestro en Ciencias con Especialidad En Sistemas
Titulo de la tesis:	ANTOLOGIA DE LENGUAJES DE PROGRAMACION
Campo o rama profesional:	Ing. en Sistemas Computacionales
Lugar y fecha de nacimiento:	Cd. Nueva Rosita, Coahuila 29 de Septiembre de 1973
Nombre del los padres:	Sr. Santos Ramírez Hernández Sra. Ernestina Hernández de Ramírez
Escuelas y Universidades	Instituto Tecnológico de Saltillo
de las que se graduó:	Cd. Saltillo, Coahuila (1990-1994)
Título Obtenido:	Ing. En Sistemas Computacionales
Experiencia profesional y organizaciones profesionales a las que pertenece	1) Instituto Tecnológico de Estudios Superiores de la Región Carbonífera, docente del mismo y apoyo en el Departamento de Desarrollo Académico. Ingrese el 01 de Febrero del 2000 a la Fecha. 2) Secretaría de Finanzas del Estado de Coahuila. Analista de sistemas ingrese el 01 de Septiembre de 1998 a 1 de Febrero del 2000 3) Agencia Aduanal Eduardo Castañeda González. Analista de Sistemas ingrese el 01 de Agosto de 1997 a 31 de Agosto de 1998. 4) Instituto Tecnológico de Estudios Superiores de la Región Carbonífera, docente del mismo. Ingrese el 01 de Enero de 1995 a Julio de 1997.