

23
2ej



UNIVERSIDAD NACIONAL AUTONOMA
DE MEXICO

FACULTAD DE INGENIERIA

SISTEMA TUTOR DINAMICO PARA LA ENSEÑANZA
DEL ALGEBRA, GEOMETRIA ANALITICA Y CALCULO
DIFERENCIAL DE UNA VARIABLE

T E S I S

QUE PARA OBTENER EL TITULO DE:
INGENIERO EN COMPUTACION
P R E S E N T A :
RAFAEL ARTURO CRUZ CRUZ

DIRECTOR DE TESIS:
FIS. RAYMUNDO H. RANGEL GUTIERREZ

MEXICO, D. F.

1983



TESIS CON
FALLA DE ORIGEN



UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

DERECHOS RESERVADOS © PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis está protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

INDICE

	pag.
INTRODUCCION	1
Capítulo I	
Características de los paquetes educativos y de los sistemas de computación.	15
Requisitos didácticos	17
Principios didácticos	19
Requisitos de operación	20
Requisitos de los paquetes educativos	23
Documentación	26
Sistemas de computación	26
Ciclo de vida de un sistema de programación	29
Capítulo II	
Análisis de requerimientos	32
Análisis	33
Etapas de análisis	33
Principios fundamentales de los métodos de análisis	35
Análisis orientado a objetos	37
Construcción de prototipos de software	39
Métodos de análisis	45
Método de análisis orientado al flujo de datos	48
Método de análisis orientado a la estructura de datos	54
Alternativas de solución	59
Álgebra computacional	60
Sistema de álgebra computacional "MAT-UNAM"	65
Elección de la alternativa más viable	69
Selección del lenguaje de programación	70
Capítulo III	
Diseño del sistema	73
Surgimiento del diseño orientado a objetos	74
Pasos para el diseño orientado a objetos	79
Un vistazo a Smalltalk y C++	84
Objetos	87
Polimorfismo	87
Herencia	88
Constructores y destructores	89
Diseño de los módulos del sistema MAT-UNAM	90
Módulo Editor	90
Módulo Evaluar	94
Módulo Graficar	101
Módulo Tutor	102

Capítulo IV	
Programación y pruebas	105
Programación	105
Objetivo del compilador	108
Objetivo del editor	119
Pruebas al sistema MAT-UNAM	131
Capítulo V	
Conclusiones	134
Bibliografía	138
Apendice A	
Manual de Usuario	A-1
Apendice B	
Diagrama de flujo de datos del sistema MAT-UNAM	B-1
Apendice C	
Gramática del compilador de MAT-UNAM	C-1
Apendice D	
Operaciones con polinomios	D-1

INTRODUCCION

En la actualidad la educación juega un papel de primordial importancia, sobre todo en países como el nuestro. Es por ello que la formación de los estudiantes de ingeniería debe apoyarse en la tecnología. Así pues, una adecuada capacitación sobre el uso de la computadora, les permitirá tener a ésta como una herramienta primordial.

Por esta razón, el sistema Mat-UNAM, se crea con la finalidad de proporcionar una herramienta de computadora para el estudiante de álgebra y cálculo diferencial en una variable.

El conjunto de elementos humanos, materiales, técnicos, y metodológicos que tienen como finalidad apoyar la educación en alguna disciplina o área, forma parte del proceso de enseñanza-aprendizaje.

En los medios de comunicación se ha demostrado que la retención de un mensaje aumenta cuando se emiten simultáneamente señales tanto auditivas como visuales. Por lo que estas características son aplicadas en el sistema Mat-UNAM.

La computadora tiene la posibilidad de conjuntarlas, por lo que resulta una herramienta efectiva que suscita el interés en el estudiante. Tiene la posibilidad de reproducir experiencias reales o

simuladas. Facilita la creatividad, y tiene la capacidad de conservar la atención del estudiante. Todo ello provoca un impacto emotivo que facilita la comprensión de los mensajes educativos.

Cuando se emplea como elemento para facilitar la autoenseñanza, al propiciar la individualización del proceso, tiene la ventaja de que el alumno participa activamente, y que la velocidad, el esfuerzo y retroalimentación del aprendizaje pueden regularse a través de la interacción tutorial con la máquina.

Otras ventajas de las computadoras dentro del proceso educativo, consisten en mejorar las habilidades de escritura, de lectura y el razonamiento lógico-matemático.

Además, la computadora es útil como instrumento de cálculo, para el aprendizaje de asignaturas apoyadas en programas que ilustran aplicaciones prácticas, de ejercicios teóricos y de laboratorio, o bien, mediante programas simuladores y tutoriales, como el caso del sistema Mat-UNAM.

Asimismo, ha permitido avances en la docencia de las disciplinas donde el procesamiento de datos, las representaciones gráficas, la implantación de modelos matemáticos y el manejo de sistemas de información son necesarios.

Sin duda alguna, un elemento clave para la introducción de la computadora como auxiliar de la educación es el profesor mismo, que debe percibir claramente el papel y valor de esta tecnología en la docencia, por lo que deben tener una preparación adecuada para su uso, a fin de evitar frustraciones en los proyectos.

Las ventajas económicas y de soporte que ofrecen actualmente las microcomputadoras, en su aplicación a la educación, se ven frenadas por la situación financiera por la que atraviesa el país; esto ha conducido a que las escuelas no cuenten con los equipos suficientes para que los maestros los utilicen en la clase y para que la interacción individual de los alumnos con la máquina en los laboratorios sea efectiva.

Por otra parte, se han presentado rechazos a las computadoras por actitudes conservadoras de tipo institucional o del personal académico, ya sea por temor o como una resistencia al cambio.

También se han presentado problemas de imprecisión y falsas expectativas en algunas instituciones educativas, que habiendo adquirido equipo de cómputo, no cuentan con un programa educativo formal para su utilización; esta situación se agrava por la falta de paquetes educativos y de entrenamiento de los profesores.

Por lo anterior, muchas instituciones educativas no cuentan con la infraestructura de cómputo necesaria ni con el personal académico capacitado en uso. lo que conduce a que muchos egresados de estudios profesionales no tengan preparación en este campo y se encuentran incompetentes en un mundo cada vez más informatizado.

En particular en el caso de la enseñanza de la ingeniería, además de los problemas ya señalados, se ha detectado que el uso de la computadora se ha orientado al aprendizaje de lenguajes que para ciertas especialidades no son los idóneos y, por otra parte, que en muchos casos la materia en que esto se enseña se encuentra aislada, desvinculada de las otras asignaturas que componen el plan de estudios de las carreras.

Además, no siempre se capacita al estudiante para el desarrollo de programas y sistemas, sino a que conozca y aplique los programas comerciales a la solución de diversos problemas, sin conocer los modelos matemáticos involucrados; es decir, se manejan como "cajas negras".

En muchas escuelas de ingeniería se ha observado el desconocimiento de las técnicas de diseño, así como el hecho de que el personal académico no cuenta con programas de capacitación en cómputo, ni con programas de desarrollo de paquetes educativos; esto

ocasiona que los profesores no utilicen la simulación por computadora como medio para representar y reproducir fenómenos que auxilien a los alumnos en el desarrollo de experiencias y habilidades que involucren situaciones complejas, ya que no cuentan con programas tutores y sistemas expertos.

El estudiante de ingeniería requiere de una formación propia de su profesión y, para ello, la matemática desarrolla un papel trascendental. El estudiante debe estudiar la matemática con formalidad para luego, en su vida profesional, *extrapolar* la formación que adquirió.

Pero teniendo presente, que el conocimiento matemático el ingeniero lo adquiere no sólo en el estudio, sino también en la experiencia y la práctica, y que para adquirirlo en estas dos últimas, es imprescindible que su estudio sobre la matemática en la escuela de ingeniería haya sido con formalidad.

Por supuesto, siempre debe buscarse el nivel de profundidad propio de las carreras de ingeniería. El enfoque y estudio del cálculo es, obviamente, diferente en Ciencias que en Ingeniería y que en Contaduría y Administración. De igual manera para las ecuaciones diferenciales y para cualquier otro ramo de la matemática.

Pero, si bien es cierto que la profundidad y extensión con las que se estudian las matemáticas deben ser diferentes, dependiendo de la Facultad en la cual se estudien, siempre deberán presentarse con claridad y formalidad. El lenguaje sencillo, e incluso llano y familiar, no está reñido con la formalidad.

El profesor debe evitar el uso de expresiones incorrectas tanto orales como escritas, y corregir al estudiante cuando éste las emplea. El maestro de matemáticas debe usar el lenguaje correcto y formal al mismo tiempo que claro y sencillo. Basta recordar lo dicho por Courant y Robbins:

'' La matemática, como una expresión de la mente humana, refleja la voluntad activa, la razón contemplativa y el deseo de perfección estética''

Desde épocas remotas el hombre se ha enfrentado a diferentes tipos de problemas, en los cuales se requería una serie de cálculos para llegar a la solución. Por esta razón, el hombre siempre se ha preocupado por inventar técnicas que faciliten la solución de este tipo de problemas.

Las primeras técnicas utilizadas fueron las de conteo, dentro de estas tenemos:

- 1) Cuento con los dedos u otros objetos (piedras, varas, etc.).
- 2) El ábaco (China 2.600 A.C.).
- 3) Tabla de logaritmos (Escocia, Jhon Napier 1614).
- 4) Regla de cálculo (Inglaterra, William Oughtred 1632).
- 5) Máquina de pascal (Francia, Blaise Pascal 1642).
- 6) Máquina de Jacquard (Joseph Marie J. 1801).
- 7) Máquina de diferencias (Charles Babbage 1812).
- 8) Calculadoras de tarjeta perforada (E.U.A. Herman Hollerith).

Una aplicación de la computadora es la científica. Este tipo de aplicaciones se caracterizan principalmente por la manipulación de números y arreglos de estos, utilizan principios matemáticos y estadísticos como base de sus algoritmos.

Estos algoritmos tratan problemas tales como: exámenes estadísticos, programación lineal, análisis de regresión y aproximaciones numéricas para la solución de ecuaciones diferenciales e integrales, así como una diversa gama de problemas ingenieriles.

Los problemas científicos suelen tener una considerable complejidad matemática, razón por la cual, los programadores deberán conocer los principios matemáticos necesarios para abordar correctamente los problemas o hacer refinamientos, pues solo así se tendrán soluciones adecuadas.

Los problemas científicos requieren en la mayoría de los casos, de mayor trabajo por parte del procesador central de una computadora, que de los dispositivos de entrada/salida, esto es debido a que la solución es en base a cálculos, con un mínimo de informes durante el proceso.

La problemática de las matemáticas.

Los problemas de la enseñanza y aprendizaje de las matemáticas son de sobra conocidos, recientemente han adquirido otras características como: un aspecto masivo, su amplitud de alcance ya que cada vez son estudiadas en más carreras y estudios técnicos y la incorporación de nuevas ramas a los programas de estudio, etcétera.

En los últimos treinta años ha habido diversos intentos de mejorar la enseñanza de las matemáticas y subsanar algunas de sus principales deficiencias que, en términos generales, han tenido resultados muy inferiores a los programados.

Podemos afirmar que a grandes rasgos, estos intentos han seguido el esquema tradicional de finalidades, medios y "estrategias de resolución" de problemas educativos y académicos que prevalece en las universidades y centros de actividad académica.

Esto es, pequeños grupos de especialistas o expertos en investigación educativa investigan los problemas y generan soluciones que son llevadas a propuestas a la gran masa de profesores de matemáticas a quienes corresponde incorporarlas a la práctica.

Hasta la fecha ha producido soluciones raquíticas a los grandes problemas de la enseñanza y aprendizaje de las matemáticas. Los principios o bases prácticas y teóricas en que descansa este esquema están en contradicción total con principios básicos de carácter pedagógico, psicológico, matemático y sociológico.

La comprensión de las matemáticas demanda esfuerzos intelectuales particularmente intensos del estudiante que no sólo no son apoyados por el contexto educativo (ni el cotidiano) sino que en general son obstaculizados por éste.

Quienes enfrentan los problemas son los profesores y los estudiantes, y son éstos los que conjuntamente durante la práctica deben ir generando alternativas de solución.

Un esfuerzo serio y a fondo para resolver los problemas de la enseñanza de las matemáticas requiere indispensablemente tanto de la incorporación masiva de los actuales profesores al estudio de los

problemas de la enseñanza de las matemáticas como de una nueva preparación de los futuros profesores, y aplicación de la tecnología con que se cuenta, como la computadora.

Las escuelas superiores de matemáticas en general tienen una posición muy ventajosa en términos de recursos académicos, y económicos, autonomía, influencia en la enseñanza de las matemáticas de ciertos niveles, etcétera, que pueden y deben aprovechar para promover un trabajo amplio y sistemático en la problemática de la enseñanza de las matemáticas.

Al enfrentarnos con el problema cada vez más agudo que presenta la enseñanza de la matemática, debido, entre otras causas, a las exigencias que imponen los avances científicos y tecnológicos, debemos tener presente los siguientes hechos irrefutables:

- 1.- La matemática es detestada por la mayoría de los estudiantes.
- 2.- El nivel matemático de los alumnos que cumplieron "satisfactoriamente" las exigencias de contenidos matemáticos de un ciclo escolar, es bajo o nulo. Esto se comprueba cuando deben aplicarlo en distintas situaciones concretas.

FUNDAMENTOS.

Al abordar el problema relacionado con las metodologías especiales de la enseñanza, debemos solucionar previamente ciertos aspectos fundamentales:

- Precisar el objetivo de la asignatura.
- Considerar las leyes que rigen la elaboración del conocimiento.
- Determinar los contenidos que abarca.

Los avances en distintos campos del saber humano y sus aplicaciones, imponen una adecuación de contenidos en las distintas asignaturas; pero lo que adquiere mayor trascendencia es la forma de presentación de dichos contenidos en la búsqueda de un aprendizaje efectivo.

Además proporcionar al alumno instrumentos de aplicación mecanizada, para lograr desarrollar funciones que favorezcan un pensamiento lógico.

El alumno debe tener el dominio de la matemática rigurosa, pero por caminos que alteren su proceso evolutivo, sin presiones contraproducentes sobre su mentalidad en desarrollo.

El método, el camino recorrido por la ciencia matemática en su formación, no es el deductivo puro (propio de la matemática ya

estructurada), que es el que queda como único válido en la mente del matemático y es así llevado al campo de la enseñanza.

Si el matemático utiliza esquemas simples como punto de partida para configurar estructuras más complejas en el camino del desarrollo de esta ciencia, no podemos aceptar que al alumno se le impongan conceptos totalmente elaborados, relaciones no verificadas por él mismo o procesos deductivos puros, cuando en realidad el camino en su sistematización es el inverso.

Por lo tanto, una metodología adecuada debe ajustarse a las características siguientes:

- a) Debe seguir un proceso primeramente inductivo y luego deductivo.
- b) Debe presentar una secuencia gradual, desde el nivel más concreto posible al más abstracto.
- c) Debe ser activa; el alumno elabora los conceptos recurriendo a la acción motora (manipuleo de material concreto) o a la acción intelectual.
- d) Debe dinamizar la presentación de los contenidos, como incentivos de la actividad del alumno.

Objetivo general.

Objetivo:

Crear un sistema tutor en un equipo PC con retroalimentación al usuario final, mostrando paso a paso la solución de un problema propuesto por el mismo sistema, el profesor o el alumno.

Requerimientos del sistema:

- Sistema de matemáticas enfocado al álgebra básica.
- El sistema debe contemplar las funciones básicas de un sistema de álgebra computacional.
- El sistema debe ser de fácil manejo.
- El sistema debe ser capaz de obtener gráficas.

CAPÍTULO I

CARACTERÍSTICAS DE LOS PAQUETES EDUCATIVOS Y DE LOS SISTEMAS DE COMPUTACION

Los programas de computación educativa tienen como objetivo principal, incrementar la productividad en el proceso de capacitación de conocimientos por parte de los alumnos, y herramienta adicional de apoyo al profesor.

Las características que deben brindar los programas de computación educativa son:

- Apoyo didáctico al profesor.
- Interacción de los alumnos con la computadora.
- Operación ante el grupo.
- Operación complementaria en laboratorio.

Existe una serie de normas generales que deben seguirse al desarrollar los paquetes educativos para conservar una homogeneidad entre ellos, así pues, se tienen los siguientes requisitos:

- DIDACTICOS.
- DE OPERACION
- DE PROGRAMACION.
- DOCUMENTACION.

Requisitos didácticos.

En cuanto a su manejo, deben contener elementos complementarios para que su operación sea sencilla y puedan ser utilizados por los estudiantes en forma alternativa.

Por lo que respecta a los temas tratados, es indispensable que incluyan los elementos necesarios para lograr la integración del conocimiento.

Conjuntamente con esto, se debe dar prioridad a los temas en que tradicionalmente se ha detectado tener problemas de transmisión y entendimiento, buscando reducir los índices de reprobación. Deben desarrollarse principalmente en temas para los cuales no se cuenta con la posibilidad de apoyo para el maestro a través de otro medio didáctico.

Deben contener elementos de simulación y demostración para facilitar la explicación gráfica de los conceptos deseados y ejercicios con diferentes niveles de dificultad, para que posteriormente los alumnos puedan reforzar los conocimientos adquiridos en el salón de clase.

Para lograr todo esto, se debe hacer uso de todas las técnicas computacionales que permitan la explotación máxima de los recursos disponibles.

Para el diseño de los programas educativos debe considerarse los puntos siguientes:

- Al desarrollar el contenido del programa, los autores deben tener en cuenta el nivel de aprendizaje en que se encuentra el alumno. Por tal motivo, deben cuidar que la información guarde una secuencia lógica y conduzca a la asimilación del conocimiento, con el grado de dificultad específico para el nivel que estipule la guía de contenido o esquema temático.

- Indistintamente del tipo de materia a tratar en el programa, el autor debe considerar siempre la interacción entre los alumnos, el profesor y la computadora.

El programa dentro de su estructura, debe incluir los siguientes elementos:

- 1) Identificación: es necesario indicar el nivel al que pertenece el programa, área, materia y unidad. El título del tema a desarrollar y el objetivo que debe cumplir el tema al que se refiere el programa.
- 2) Introducción o tema de contacto: En este punto se señala el campo de aplicación del conocimiento presentado, relacionandolo en resto de los temas resaltando la importancia del mismo (concepto de integración).
- 3) Desarrollo: Siendo esta la parte más importante, debe tenerse en cuenta la necesidad de lograr captar la atención en el tema, estableciendo relaciones con la realidad que se vive

cotidianamente.

4) Evaluación: El incluir ejercicios de evaluación da al alumno la oportunidad de evaluar su aprovechamiento. Cabe recordar que existen diversas formas de evaluación, se deberá elegir aquella capaz de satisfacer los intereses que se persiguen de acuerdo al objetivo planteado.

5) Retroalimentación del conocimiento: Esto se hace mediante consulta a la microcomputadora (individual, en grupo o con el profesor), sobre él, o los resultados de los trabajos realizados.

Principios didácticos.

- Principio de intuición: Visualización u objetivación de la enseñanza.

- Principio de generalidad: Todos los temas deben partir de lo general a lo particular.

- Principio de la actividad en la enseñanza: Debe procurarse que el alumno participe de manera activa en el proceso de enseñanza-aprendizaje.

- Principio de consolidación del éxito: Darle al alumno la seguridad de que ha aprendido.

- Principio de adecuación del estudiante: Presentar el contenido conforme a las características generales de los estudiantes, en su nivel evolutivo y educativo.

Requisitos de operación.

En cuanto al modo de operación, el esquema a seguir es semejante al uso del pizarrón; en ocasiones el mismo profesor maneja el teclado y hace la explicación de resultados, en otras el alumno maneja el teclado y hace la explicación con el apoyo del maestro, en otras más puede utilizarse para evaluaciones de los mismos.

Dentro de la Instrucción Asistida por Computadora (ICA), se manejan básicamente seis formas de transmisión del conocimiento, estas son:

1) Ejercicios y prácticas.

El profesor presenta su clase sobre un tema y los estudiantes utilizan el programa de la computadora para reafirmar los conocimientos derivados de la lección. Se da preferencia a este uso cuando el tema a tratar requiere de habilidades que se desarrollan solamente con base en ejercicios de repetición, en los que las condiciones cambian y el grado de dificultad es creciente o decreciente.

2) Recuperación de la información.

El programa de computadora es preparado previamente con un banco de datos, que es examinado por los alumnos con base en las indicaciones del profesor. Esta técnica es útil cuando el tiempo del que dispone el profesor es breve y el número de estudiantes es grande.

Los programas de esta naturaleza pueden adquirir varias formas, ya sea como series de preguntas y respuestas en las que se ofrece una sucesión de cuestionamientos alternativos, o bien, como un simple banco de datos que el estudiante puede consultar con facilidad.

3) Video interactivo.

El video interactivo presenta imágenes similares a las de un filme o video cinta. El programa posee un archivo de datos que el alumno manipula de acuerdo a las reglas y comandos previamente establecidos, para poder visualizar la información deseada.

4) Juegos.

Esta es probablemente la más conocida y comercial actividad educativa en la que están involucradas las microcomputadoras. La cantidad de paquetes disponibles es grande y continua en aumento. El objetivo es transformar una tarea a un juego en el que la computadora es, o simula ser un oponente, y el estudiante tiene siempre una alta posibilidad de perder.

Es importante señalar que los juegos deben tomar una característica más parecida a juegos de agilidad mental que de agilidad motriz.

5) Demostraciones.

Para este uso, la computadora se convierte en un modelo de situaciones complejas o de objetos. La falta de recursos o de material didáctico es suplida por la computadora, la cual llega a funcionar como un laboratorio en el que se experimenta visualmente con sustancias o animales y se estudian la reacciones.

6) Simulaciones.

Indudablemente, éste es el más ambicioso de los usos de las computadoras en el salón de clase.

La simulación suele ser utilizada para representar un rango enorme de posibilidades, desde el vuelo de un avión, hasta el comportamiento de un huracán.

A la vez, la representación puede ser totalmente abstracta y proposicional, o concreta y visual, según las aptitudes y preferencias del grupo y de las características del tema por tratar.

Requisitos de los paquetes educativos.

Tratando de poseer un total control sobre el material didáctico desarrollado, se siguen una serie de normas de programación que dan estructura y presentación homogénea a los paquetes educativos. Estas normas se describen en los siguientes puntos:

- 1) Para el caso de llamar subrutinas que requieran de parámetros, deberán señalarse las variables involucradas.
- 2) Los comentarios serán breves y concisos, ya que esta información consume espacio en memoria.
- 3) Los programas deben desarrollarse en una secuencia de rutinas cerradas, minimizando el uso de variables globales.
- 4) La elaboración de todo programa debe ser estructural o modular, de tal forma que tenga un programa principal y de éste, sean llamados los subprogramas (o rutinas) que se requieran.
- 5) Los nombres de variables no deben exceder de 3 caracteres (en caso de programar en BASIC), y deben dar una idea de la función de la variable.
- 6) Se utilizarán varias instrucciones en cada línea de código, a fin de ahorrar memoria.

- 7) Se debe contar con un subprograma que contenga definiciones y conceptos necesarios para lograr un buen aprendizaje del tema desarrollado en el paquete. Este subprograma se conoce con el nombre de "archivo", al cual se puede tener acceso en cualquier momento que se desee para ver la información que contienen.

- 8) Al validar una respuesta a ejercicios, se deben aceptar solamente los valores dentro de los rangos de las opciones válidas. Cualquier otra tecla oprimida debe ser ignorada o pedir que se vuelva a teclear la respuesta.

- 9) Para avanzar de una pantalla a otra, siempre será utilizada la tecla ENTER.

- 10) Se debe contar además con una subrutina conocida como "barra de operación" que facilite el uso de los paquetes didácticos, la cual debe presentar las siguientes opciones:
 - Pasar a la siguiente pantalla.
 - Regresar a la pantalla anterior.
 - Pasar directamente al menú o contenido.
 - Pasar a un número de pantalla deseado.
 - Pasar a la parte de "archivos".
 - Poder solicitar información relacionada con el funcionamiento de esta barra en el momento que se desee.

- La barra de operación tiene diferente presentación de acuerdo al modo gráfico necesario para mostrar un pantalla (para BASIC modo 3 para textos o modo 4 para dibujos).
- Cuando se presenta la parte de ejercicios, se sustituye la opción de regresar a la pantalla anterior por la de evaluación de ejercicios.
- Todas las opciones anteriores pueden ser elegidas en el momento que se desee cuando se presenten las diferentes pantallas que integran un paquete.

11) En el caso de que un paquete tenga que ser particionado en más de un programa por razones de la capacidad de memoria, se procedera de la siguiente forma:

- Se dividirá en base al contenido del programa, dejando partes completas en cada programa.
- Cada programa tendrá la presentación y el menú de esa parte, tomando en cuenta que forman parte de un tema en particular.
- Cada programa debe contar con un punto de ejercicios así como otro para finalizar la ejecución del mismo. además de los puntos que conforman la parte de que se trate.

12) El nombre del programa estará constituido por ocho caracteres, en la siguiente forma:

- Las primeras dos posiciones identifican el área del tema del programa.

- La tercera y cuarta posiciones indican el grado y la unidad a la que pertenece el programa.
- La quinta y sexta posición es para indicar el número del tema que cubre el programa.
- La séptima posición es para indicar el número de parte cuando el tema se fraccione.
- La octava posición especifica el número de versión del programa.

Documentación.

La documentación necesaria para que los programas de computación educativa sean aceptados, consiste en:

- 1) Programa fuente grabado en cassette o disco.
- 2) Manual de operación del programa que describe el proceso a seguir para su funcionamiento, señalando las partes en que fue dividido el tema y las respuestas a los ejercicios planteados.
- 3) Manual del profesor, incluyendo las especificaciones pertinentes en relación al programa, para su correcto manejo. Además contiene sugerencias para que el profesor haga comentarios a los alumnos sobre el tema tratado.

Sistemas de computación.

Como consecuencia de la necesidad de planificar, operar y diseñar sistemas cada día más complejos que solucionen los grandes problemas sociotécnicos en los medios académicos e industriales, ha surgido una nueva filosofía o metodología llamada Ingeniería de

Programación.

El objeto final de este trabajo es llevar a cabo el desarrollo de un sistema, sin embargo, no se puede llegar a éste sin antes haber entendido el problema, sin haber hecho un estudio y una planeación y sin haber considerado otros aspectos que se incluyen en lo que se denomina metodología de desarrollo.

El manejo de la información por medio de computadoras, se hace día a día más frecuente, sin embargo el diseñar sistemas adecuados y eficientes para la manipulación de dicha información no resulta fácil.

En ocasiones el analista se enfrenta a problemas de diseño, el programador a problemas de codificación y el usuario se enfrenta con un sistema deficiente, eso se debe en gran medida a la falta de uso de una técnica bien definida para el diseño del sistema.

Las técnicas empleadas son diversas, pero todas pretenden llegar a lo mismo: desarrollar sistemas que sean eficientes, fácilmente implementables y bien documentados.

Dichas técnicas se agrupan en lo que se conoce como Ingeniería de Software o Ingeniería de Programación.

La Ingeniería de Programación es una disciplina que hasta la década de los sesentas aún no se había establecido. Surgió como una solución a lo que se conoce como la Crisis del Software.

La problemática principal que se detectó en dicha crisis se puede definir en los siguientes puntos:

- + Los costos se incrementaban en forma exponencial.
- + Los proyectos no se terminaban a tiempo ni con el presupuesto programado.
- + El mantenimiento del sistema absorbe la mayor parte de recursos de la gente de desarrollo.

Los problemas anteriores conllevan a la generación de problemas asociados de los que se destacan los siguientes:

- + Insatisfacción del usuario con el sistema terminado.
- + Calidad dudosa del software.
- + Dificultad de mantenimiento del software actual.

Uno de los factores que se tomaron como referencia para detectar la crisis, fué la comparación de costos entre el hardware y el software, mientras que en uno disminuían, en otro aumentaban considerablemente.

Lo que pretende la ingeniería de software, es establecer métodos y técnicas que lleven paso a paso a la realización de sistemas de cómputo eficaces, eficientes, confiables, transportables y rentables. Por otro lado, define lo que es el ciclo de vida de un proyecto de software.

Ciclo de vida de un Sistema de Programación.

Existe un número considerable de metodologías de desarrollo, pero en todas se puede distinguir las siguientes fases:

Estudio del sistema.

Se refiere a una revisión general del sistema actual, de la que se deriva la detección y definición de necesidades, se plantean diferentes alternativas para satisfacer las necesidades. Se realiza un estudio de factibilidad que incluye una revisión de los recursos para determinar si es factible su utilización y un análisis beneficio/costo.

El estudio de factibilidad se hace generalmente para todas las alternativas propuestas, ya que de éste se concluye la más viable. Cabe aclarar que de este estudio se puede concluir que la realización del sistema es o no factible, y ya no se continuaría con las siguientes etapas.

Planeación.

Una vez que se definió que se necesita el nuevo sistema y que es factible llevarlo a cabo, se hace una planeación de cómo se desarrollará el mismo. Primero se definen los alcances del proyecto. Se determina también la disposición de los recursos cronológicamente y se establecen mecanismos de supervisión y control de avance.

Con respecto a las herramientas de control del proyecto, se tienen las opciones de: Gráficas de Gant, las cuales consisten de un

reporte en forma de tabla donde se incluye la información del avance, responsable, actividades, etc.

La otra opción es la realización de la Ruta Crítica, la cuál consiste en una red donde los nodos denotan las actividades a realizarse, la ruta crítica suele utilizarse en proyectos muy grandes donde se requiere gran coordinación.

Análisis y diseño.

En esta etapa, se determinan los datos que va a manejar el sistema, la localización de los mismos (archivos que se van a utilizar), los procesos que se requieren y la forma en que se comunicarán. Con los resultados de esta fase, se puede definir el lenguaje más óptimo para que se haga el desarrollo del sistema.

Programación y pruebas.

Se refiere a la codificación de los programas en un lenguaje de computación y a su depuración hasta dejarlos en buen funcionamiento.

Liberación, Instalación y Documentación.

Esta etapa es en la que se presenta el sistema terminado al usuario con la documentación necesaria para su operación y mantenimiento.

Mantenimiento.

Son los cambios, reducciones o ampliaciones que el sistema vaya requiriendo durante su vida operable.

Deseso del sistema.

Ocurre cuando se requiere de un nuevo sistema que mejore al anterior. Esto ocurre cuando la organización donde operaba el sistema cambia de políticas, cuando el sistema se vuelve obsoleto por equipo de hardware o por que se requiere de otro tipo de procesos que no son compatibles con los que se plantearon al principio del sistema en deseso. El nuevo sistema pasará por todas las etapas citadas.

Las etapas anteriores son las correspondientes al ciclo de vida de un sistema computarizado. A continuación se presentará una metodología que en la actualidad es la más utilizada por ser una de las más completas.

CAPITULO II

ANALISIS DE REQUERIMIENTOS

Análisis.

El análisis y especificación de de requerimientos no es una tarea fácil, puesto que el contenido de comunicación es muy alto; y existen constantes cambios por mala interpretación o falta de información.

Para realizar bien el desarrollo de software es esencial realizar una especificación completa de los requerimientos de los mismos. Independientemente de lo bien diseñado o codificado que esté un programa.

El análisis de requerimetros plantea la asignación de software a nivel de sistema y el diseño de programas. Además facilita especificar la función y comportamiento de los programas, indicar la interfaz con otros elementos del sistema y establecer las relaciones de diseño que debe cumplir el programa.

Etapas del análisis.

El análisis esta constituido por cuatro etapas que son:

- Reconocimiento del problema.
- Evaluación y síntesis.
- Especificación.
- Revisión.

Reconocimiento del problema : El analista debe establecer contacto con el equipo técnico, el usuario y con la empresa que vaya a desarrollar el software. Su principal objetivo es reconocer los elementos básicos del programa tal como lo percibe el usuario final.

Evaluación y síntesis : El analista debe evaluar el flujo y estructura de la información, refinar en detalle todas las funciones del programa , establecer las características de intreface del sistema y describir las relaciones de diseño.

Especificación : Una vez que se hayan descrito las funciones básicas, comportamiento, intreface e información, se especifican los criterios de validación para demostrar una comprensión de una correcta implementación de los programas. Para definir las características y atributos del software se escribe una especificación de requerimientos formal. Además, en los casos en que se desarrolle un prototipo se realiza un manual de usuario preliminar.

Revisión : La revisión de los requerimientos casi siempre produce modificaciones en la función, comportamiento, representación de la información, criterios de validación. La revisión es llevada a cabo por el técnico y el usuario, basandose en especificación y manual de usuario en su caso.

Principios fundamentales de los métodos de análisis.

Cada método de análisis tiene su notación y punto de vista, pero existe un conjunto de principios fundamentales para todos ellos, dichos principios son los siguientes:

- El dominio de la información, así como el dominio funcional de un problema debe ser representado y comprendido.
- El problema debe subdividirse de forma que se descubran los detalles de una manera progresiva (o jerárquica).
- Deben de desarrollarse las representaciones lógicas y físicas del sistema.

Dominio de la información.

El dominio de la información contiene tres visiones diferentes de los datos:

- El flujo de la información.
- El contenido de la información.
- La estructura de la información.

Flujo de la información.

El flujo de la información representa la manera en que los datos cambian conforme pasan a través de un sistema.

Contenido de la información.

El contenido de la información representa los elementos de datos individuales que componen otros elementos mayores de información.

Estructura de la información.

La estructura de la información representa la organización lógica de los distintos elementos de los datos. La organización lógica podría ser una arreglo de n dimensiones, un árbol jerárquico, etc., también, que relación existe entre un estructura de información y otra.

División del problema.

Normalmente los problemas son demasiado grandes y complejos para ser comprendidos como un todo. Por esta razón, los problemas se dividen en partes que puedan ser comprendidas fácilmente, y establecer interfaces entre las partes, de forma que se realice la función global.

Durante la división del problema se establece una representación jerárquica de la función o información y luego se parte el elemento superior mediante:

- Incrementando los detalles, moviendolos verticalmente en la jerarquía, o
- Descomponiendo funcionalmente el problema, moviendolos horizontalmente en jerarquía.

Representaciones lógicas y físicas.

La visión lógica de los requerimientos del software presenta las funciones que han de realizarse y la información que ha de procesarse independientemente de los detalles de implementación. Una representación lógica es un fundamento esencial para el diseño.

La visión física presenta una manifestación del mundo real de las funciones de procesamiento y las estructuras de información. En algunos casos se desarrolla una representación física como el primer paso del diseño de software.

Análisis orientado a objetos.

Las técnicas de ingeniería software orientadas a objetos han generado un amplio interés en los últimos años. En este contexto un objeto puede verse como un elemento de información y una operación, como un proceso o función que se aplica a uno o más objetos.

El análisis orientado a objetos nos proporciona un mecanismo sencillo, pero poderoso, para identificar objetos y operaciones. El método de análisis orientado a objetos puede describirse de la siguiente forma:

1.- El software asignado se describe usando una estrategia informal. La estrategia no es más que una descripción en lenguaje natural de la solución del problema que hay que resolver, mediante el software representado a un nivel consistente de detalle. La estrategia informal puede ser establecida en forma de párrafos sencillos, gramaticalmente correctos.

2.- Los objetos se determinan subrayando cada nombre e introduciéndolo en una tabla sencilla. Deben anotarse los sinónimos. Si se requiere que el objeto se implemente como una solución, entonces es parte del espacio de solución; en otros casos, si un objeto es necesario sólo para describir una solución, es parte del espacio del problema.

3.- Los atributos de los objetos se identifican subrayando todos los adjetivos y luego asociándolos con sus objetos respectivos.

4.- Las operaciones se determinan subrayando todos los verbos, frases y predicados y relacionando cada operación con el objeto apropiado.

5.- Los atributos de las operaciones se identifican subrayando todos los adverbios y luego asociándolos con sus operaciones respectivas.

Construcción de prototipos de software.

El análisis debe ser conducido independientemente de la aplicación de la ingeniería de software. En algunos casos es posible aplicar los principios de análisis fundamental y derivar a una especificación en papel del software desde el cual pueda desarrollarse un diseño.

Hay circunstancias que requieren la construcción de un prototipo al comienzo del análisis, puesto que el modelo es el único medio mediante el que los requerimientos pueden ser derivados efectivamente. A continuación se presentan los pasos para la construcción de prototipos.

Paso 1. Evaluar la petición del software y determinar si el programa a desarrollar es un buen candidato para construir un prototipo.

Paso 2. Dado un proyecto candidato aceptable, el analista desarrolla una representación abreviada de los requerimientos.

Paso 3. Después de que se haya revisado la representación de los requerimientos, se crea un conjunto de especificaciones de diseño abreviadas para el prototipo.

Paso 4. El software del prototipo se crea, prueba y refina.

Paso 5. Una vez que el prototipo ha sido probado, se presenta al cliente, el cual, "conduce la prueba" de la aplicación y sugiere modificaciones.

Paso 6. Los pasos 4 y 5 se repiten iterativamente hasta que todos los requerimientos estén formalizados o hasta que el prototipo haya evolucionado hacia un sistema de producción.

Para que la construcción de prototipos de software sea efectiva, un prototipo debe desarrollarse rápidamente, de forma que el cliente pueda comprobar los resultados y recomendar cambios. Para conseguir una construcción rápida de prototipo, existen tres clases genéricas de métodos y herramientas : Técnicas de la cuarta generación, componentes de software reusables, especificación formal y entornos de construcción de prototipo.

Componentes de software reusables.

Otro método para la construcción rápida de prototipos es ensamblar, en vez de construir, el prototipo usando un conjunto de componentes de software existente. Debe observarse que un producto existente de software puede ser usado como un prototipo para un producto nuevo mejorado y competitivo.

Especificación formal y entornos para la construcción de prototipos.

En las pasadas dos décadas se han desarrollado varios lenguajes de especificación formal para reemplazar las técnicas de especificación en lenguaje natural. Actualmente se están desarrollando entornos interactivos con las siguientes características:

- 1) facilite al analista crear interactivamente una especificación basada en un lenguaje de computación.
- 2) llame a herramientas automáticas que traduzcan las especificaciones basadas en lenguaje en código ejecutable, y
- 3) faciliten al cliente utilizar el código ejecutable del prototipo para refinar los requerimientos formales.

Las técnicas de análisis pueden conducir a una especificación en papel que contenga las descripciones gráficas y el lenguaje natural de los requerimientos del software. La construcción de prototipos conduce a una especificación ejecutable, esto es, el prototipo sirve como una representación de los requerimientos.

Principios de la especificación.

Baltzer y Goldman proponen ocho principios para una buena especificación.

Principio 1. Separar funcionalidad de implementación.

Por definición, una especificación es una descripción de lo que se desea, en vez de cómo se realiza (implementa).

Principio 2. Se necesita un lenguaje de especificación de sistemas orientado al proceso.

Tales especificaciones orientadas al proceso, que presentan un modelo del comportamiento del sistema, han sido normalmente excluidas de los lenguajes de especificación formales, pero son esenciales si han de especificarse situaciones dinámicas más complejas.

Principio 3. Una especificación debe abarcar el sistema del cual el software es una componente.

Un sistema está compuesto de componentes que interactúan. Sólo dentro del contexto del sistema completo y de la interacción entre sus partes puede ser definido el comportamiento de una componente específica.

Principio 4. Una especificación debe abarcar el entorno en el que opera el sistema.

De hecho, la única diferencia entre el sistema y su entorno es que el esfuerzo de diseño e implementación subsecuente operará exclusivamente sobre la especificación del sistema. La especificación del entorno facilita que se especifique la "interface" del sistema de la misma forma que el propio sistema, en vez de introducir otro formalismo.

Principio 5. Una especificación del sistema debe ser modelo cognitivo.

Debe describir un sistema tal como es percibido por el usuario. Los objetivos que manipula deben corresponderse con objetos reales.

Principio 6. Una especificación debe ser operacional.

La especificación debe ser completa y lo bastante formal para que pueda usarse para determinar si una implementación propuesta satisface la especificación de pruebas elegidas arbitrariamente.

Principio 7. La especificación del sistema debe ser tolerante con la incompletitud y aumentable.

Ninguna especificación puede ser siempre totalmente completa. El entorno en el que existe es demasiado complejo para ello. Una especificación es siempre un modelo -una abstracción- de alguna situación real (o imaginada). Por tanto, será incompleta.

Principio 8. Una especificación debe ser localizada y débilmente acoplada.

Como pueden ocurrir tantos cambios en la especificación, es crítico que su contenido y estructura se elija de forma que pueda realizarse esta actividad. Los principales requerimientos para tales adecuaciones son que la información dentro de la especificación, debe estar localizada de forma que cualquier pedazo de información pueda ser añadido o quitado fácilmente y la estructura reajustada automáticamente.

La especificación de requerimientos de software se produce en la culminación de la tarea de análisis. La función y comportamiento asignados al software como parte de la ingeniería de sistemas se refina estableciendo una descripción completa de la información.

Una descripción funcional detallada, una indicación de los requerimientos de rendimiento y las ligaduras de diseño, unos criterios de validación apropiados y otros datos pertinentes a los requerimientos. A continuación se especifican criterios para la especificación:

1. Introducción
2. Descripción de la información
3. Descripción fundamental
4. Criterios de validación
5. Bibliografía
6. Apéndice

La introducción describe los fines y objetivos del software, describiéndolos en el contexto del sistema.

La descripción de la información da una descripción detallada del problema que el programa debe resolver. Se requiere una descripción de cada función para resolver el problema como se presenta en la descripción funcional.

Para cada función se da una explicación del procesamiento, se establecen y justifican las ligaduras del diseño, se establecen las características del comportamiento y se incluyen uno o más diagramas para representar gráficamente la estructura global del software y la interrelación entre sus funciones y otros elementos del sistema.

La sección de la especificación de requerimientos es probablemente la más importante e irónicamente la más descuidada. La sección de criterios de validación actúa como una revisión implícita de los requerimientos de información y funcionales. Es esencial que se dedique a esta sección tiempo y atención.

La bibliografía contiene referencias a todos los documentos relativos al programa. Estos incluyen otra documentación para la fase de definición, referencias técnicas, literatura de vendedor y estándares.

El apéndice contiene información que complementa a la especificación. Datos tabulares, descripción detallada de los algoritmos, planos, graficas y otros materiales se presentan como apéndices.

Métodos de análisis.

La mayoría de los métodos de análisis son conducidos por la información. Esto es, el método suministra un mecanismo para representar el dominio de la información. Desde esta representación,

se deriva la función y se desarrollan otras características de los programas.

Los métodos de análisis combinan procedimientos sistemáticos con una notación única para analizar los dominios de la información y funcional de un problema de software; suministra un conjunto de heurísticas para subdividir el problema y define una forma de representación para los aspectos lógicos y físicos.

Los métodos de análisis ayudan al analista en la construcción de una descripción precisa e independiente de los elementos de software de un sistema.

Aunque cada método introduce una nueva notación y heurística de análisis todos los métodos tienen las siguientes características comunes:

- * Mecanismos para el análisis del dominio de la información.
- * Método de representación funcional.
- * Definición de interfaces.
- * Mecanismos para subdividir el problema.
- * Soporte de la abstracción.
- * Representación de los aspectos lógicos y físicos.

Aunque el análisis del dominio de la información se conduce de forma diferente en cada metodología, pueden reconocerse algunas guías comunes. Todos los métodos se enfocan (directa o indirectamente) al flujo de datos y al contenido o estructura de datos. En la mayoría de los casos el flujo se caracteriza en el contexto de las transformaciones (funciones) que se aplican para cambiar la entrada en la salida.

El contenido de los datos puede representarse explícitamente usando un mecanismo de diccionario o, implícitamente, enfocando primero la estructura jerárquica de los datos.

La mayoría de los métodos de análisis permiten al analista evaluar la representación física de un problema antes de derivar a la solución lógica. En general, la misma notación se utiliza para representar ambos aspectos.

Los métodos y herramientas pueden dividirse en tres amplias categorías de análisis : análisis orientado a flujo de datos, análisis orientado a estructura de datos y especificación formal basada en lenguaje.

Los métodos y herramientas de las primeras dos categorías se desarrollaron originalmente para su aplicación manual y han ido mejorándose con distintas herramientas de soporte. Las herramientas que soportan entornos de especificación formal basada en lenguajes.

han sido creadas para establecer una ingeniería de los requerimientos ayudada por la computadora.

Métodos de análisis orientados al flujo de datos.

La información se transforma como un flujo a través de un sistema. El sistema acepta una entrada de distintas formas; aplica hardware, software y elementos humanos para transformar la entrada en salida; y produce una salida en distintas formas.

La entrada puede ser una señal de control transmitida por un transductor, una serie de números escritos por un operador humano, un paquete de información transmitido por un enlace a red, etc.

La transformación puede comprender una sencilla comparación lógica, un complejo algoritmo numérico, o un método de inferencia basado en reglas de un sistema experto.

La salida puede encender un sencillo led o producir un informe de 'n' páginas. Un modelo de flujo de datos puede aplicarse a cualquier sistema independientemente del tamaño o complejidad.

Diagramas de flujos de datos.

Conforme la información se mueve a través del software, se modifica mediante una serie de transformaciones. Un diagrama de flujo de datos (DFD), es una técnica gráfica que describe el flujo de información y las transformaciones que se aplican a los datos, conforme se mueven de la entrada a la salida.

El DFD puede usarse para representar un sistema o a cualquier nivel de abstracción. De hecho, los DFD pueden particionarse en niveles que representan flujo incremental de información y detalle funcional.

Un nivel 01 de un DFD, también llamado un modelo de sistema fundamental, representa el elemento de software entero como un círculo con datos de entrada y salida, indicados con flechas hacia adentro y hacia afuera, respectivamente.

El círculo representa un proceso o transformación que se aplica a los datos y que los cambia de alguna forma. Una flecha representa uno o más elementos de datos.

Criterios para la derivación de un diagrama de flujo de datos:

- 1) El nivel 01 del diagrama de flujo de datos debe describir al software/sistema como círculo sencillo.
- 2) los archivos de entrada/salida principales deben ser anotados cuidadosamente.
- 3) todas las flechas y círculos deben estar etiquetados.
- 4) la continuidad del flujo de información debe ser mantenida.

5) cada vez debe refinarse un círculo. Mediante una clara comprensión del flujo de información a lo largo del dominio de los límites del cambio, puede hacerse una mejor preparación para futuras modificaciones o puede conducirse una modificación actual sin modificar otros elementos del sistema.

Diccionario de datos.

El diccionario de datos es una gramática que describe el contenido de los elementos de información y se define de la siguiente forma:

El diccionario de datos contiene las definiciones de todos los datos mencionados en el DFD, en una especificación del proceso y en el propio diccionario de datos. Los datos compuestos (datos que pueden ser además divididos) se definen en términos de sus componentes; los datos elementales (datos que no pueden ser divididos) se definen en términos del significado de cada uno de los valores que puede asumir. Por tanto, el diccionario de datos está compuesto de definiciones de flujo de datos, archivos (datos almacenados) y datos usados en los procesos.

La notación de un diccionario de datos, facilita al analista la representación de los datos compuestos en una de las tres formas fundamentales en que puede ser construido:

- 1) como una secuencia de elementos de datos.

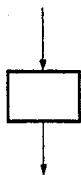
2) como una selección entre un conjunto de elementos de datos

3) como una agrupación repetida de elementos de datos.

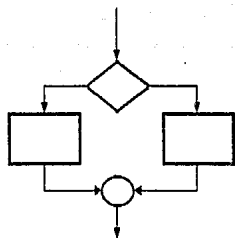
Cada entrada de un elemento de datos que se representa como parte de una secuencia, selección o repetición, puede a su vez ser otro elemento de datos compuestos, el cual necesita un posterior refinamiento dentro del diccionario.

El diccionario de datos define los elementos de información sin redundancia. Para grandes sistemas, el diccionario de datos crece rápidamente en tamaño y complejidad. De hecho es extremadamente difícil mantener un diccionario de datos manualmente.

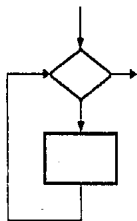
Por esta razón están disponibles varios sistemas de diccionario de datos automatizados. A continuación se muestran los símbolos que representan secuencia, selección y repetición así como, los que se utilizan para la construcción de un DFD.



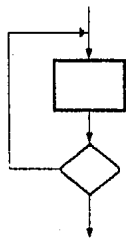
Secuencia



Selección



Repetición



52



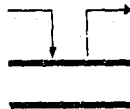
ENTIDAD EXTERNA: Una fuente de entrada al sistema,
o fuente de salida del sistema.



PROCESO: Ejecuta alguna transformación de sus datos
de entrada, produciendo sus datos de salida.



FLUJO DE DATOS: Se usa para conectar los procesos entre sí,
a las fuentes o a los suministros; La
flecha indica la dirección de transferencia
de los datos.



ALMACENAMIENTO DE DATOS: Archivo de datos, las flechas indican
las entradas y salidas de datos.

Simbolos para los Diagramas de Flujo de Datos.

El inglés estructurado (también llamado lenguaje de diseño de del programa o pseudocódigo) incorpora construcciones procedimentales básicas -secuencia, selección y repetición- junto con frases del lenguaje natural, de forma que pueden desarrollarse descripciones procedimentales precisas de las funciones representadas dentro de un DFD.

El inglés estructurado puede combinarse frecuentemente con descripciones en lenguaje natural para dar una descripción funcional completa. Durante el diseño del software se refina la descripción funcional para dar los detalles procedimentales adicionales.

Método Orientado a la Estructura de Datos.

El método de análisis orientado a la estructura de datos representa los requerimientos del software enfocándose hacia la estructura de datos en vez de al flujo de datos. Sus características son las siguientes:

- 1) Asisten al analista en la identificación de los objetos de información clave (también llamados entidades o items) y operaciones (también llamadas acciones o procesos).
- 2) Suponen que la estructura de la información es jerárquica.
- 3) Requieren que la estructura de datos se represente usando la secuencia, selección y repetición.

4) Dan un conjunto de pasos para transformar una estructura de datos jerárquica en una estructura de programa.

El método de análisis orientado a la estructura de datos proporcionan la base para el diseño de software. Siempre puede extenderse un método de análisis para que abarque el diseño arquitectural y procedimental del software.

El desarrollo de sistemas estructurados de datos (DSED), también llamado metodología de Warnier-Orr, se basa sobre el análisis del dominio de información, usando una notación para representar la jerarquía de la información usando las tres construcciones de secuencia, selección y repetición y demostró que la estructura del software podría derivarse directamente de la estructura de datos.

Orr, extendió el trabajo de Warnier para abarcar una visión algo más amplia del dominio de información, lo que llevó al Desarrollo de Sistemas Estructurados de Datos. En esta se considera el flujo de información y también sus características funcionales como jerarquías de datos.

Así pues, en vez de comenzar el análisis examinando la jerarquía de información, examina primero el contexto de la aplicación, esto es, cómo se mueven los datos entre productores y consumidores de la información.

A continuación, se establecen las funciones de aplicación con una representación, que describe los elementos de la información y el procesamiento que debe ejecutarse sobre ellos (esto es similar en concepto al diagrama de flujo de datos). El uso de este método, comprende todos los atributos del dominio de información : flujo, contenido y estructura de datos.

Usando una notación llamada diagrama de ensamblamiento de líneas (DEL), se da un mecanismo para acoplar la información y los procesos (transformaciones o funciones) que se le aplican. Un diagrama de línea se desarrolla comenzando con el único flujo de información numerado y yendo hacia atrás hasta alcanzar el primer flujo numerado.

El elemento de flujo de información es derivado combinando el ítem de información numerado precedente con el procedimiento que crea el ítem deseado.

Cada proceso es refinado desarrollando un texto del procesamiento que explica la salida, acción, frecuencia de la acción y entrada.

En este método se requiere que se construya un prototipo en papel de la salida deseada para el sistema. El prototipo identifica la salida primaria del sistema y la organización de los elementos de información que componen la salida.

Una vez que ha sido creado el prototipo, puede modelarse la jerarquía de la información usando un diagrama jerarquico o de arbol, con unas pequeñas variaciones en la notación y formato, que se deriva de los resultados de la aplicación.

La notación, que incluye entidades, línea de ensamblaje y diagramas, se utiliza para modelar los requerimientos del software desde el punto de vista lógico. También deben determinarse los requerimientos físicos como parte del análisis.

Entre los requerimientos físicos que deben considerarse están :

1) Comportamiento:

Algunas aplicaciones exigen unas ligaduras de comportamiento escritas en tiempo de ejecución. Estas pueden incluir límites definidos en tiempo de ejecución para algoritmos específicos y límites en los tiempos de respuesta para sistemas interactivos.

2) Fiabilidad:

Los procesos puede que deban ser diseñados e implementados de tal forma que aseguren un fiabilidad específica medida como tiempo medio entre fallas y disponibilidad.

3) Seguridad:

El grado con el que un sistema y el acceso a la información están protegidos puede ser una ligadura del sistema y puede tener un efecto profundo sobre la manera en que se implemente el sistema.

4) Hardware:

Las características del procesador residente deben ser tenidas en cuenta, cuando el software se acople con determinado hardware de una forma no estándar. Igualmente, deben tenerse en cuenta las características del sistema operativo.

5) Interfaces:

Los protocolos de interface con las bases de datos externas, dispositivos, redes y enlaces de comunicación pueden restringir el sistema.

Estas ligaduras físicas, cuando se acoplan con los requerimientos lógicos, proporcionan al analista un método y notación para expresar los requerimientos del software.

Alternativas de solución:

- 1) Obtener el código fuente de un paquete de algebra computacional existente y adicional la parte de tutoria
- 2) Seleccionar un paquete de algebra computacional capaz de ejecutar un programa externo al paquete.
- 3) Crear un paquete completamente nuevo con las características básicas de un paquete de algebra computacional, más un parte denominada tutoria.

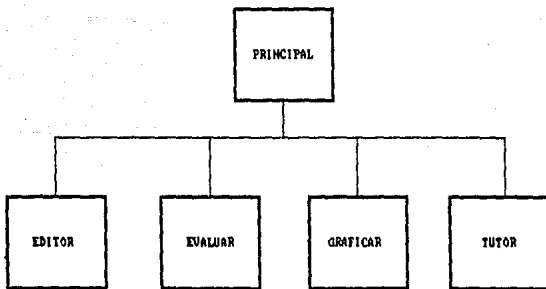


Diagrama a bloques del sistema propuesto.

Algebra computacional.

El nombre de esta disciplina ha ido cambiando desde cálculo algebraico y simbólico; manipulación algebraica y simbólica, hasta finalmente llegar a llamarse algebra computacional (COMPUTER ALGEBRA).

Al mismo tiempo las sociedades se fueron formando, juntos investigadores y usuarios de esta disciplina. La más amplia organización mundial es SIGSAM (Special Interest Group in Symbolic and Algebraic Manipulations) miembro del grupo ACM (Association for Computing Machinery), la cual organiza los congresos SYMSAC y EUROSAM, y publica el boletín SIGSAM.

El grupo europeo es llamado SAME (Symbolic and Algebraic Manipulations in Europe), organiza los congresos EUROCAM y EUROSAM.

Los investigadores franceses formaron CNRS (Centre National de la Recherche scientifique), el GRECO (Groupe de REcherches COordonnées) de algebra computacional, y desde 1985 la publicación de la revista "Symbolic Computation".

Sistemas de algebra computacional.

Hay una gran cantidad de ellos, pero de los más representativos se encuentran los siguientes:

-MACSYMA: El más desarrollado, pero desafortunadamente disponible solo en unas cuantas computadoras.

- REDUCE: El más ampliamente difundido en todos los sistemas grandes.

- muMATH: El sistema disponible para microcomputadoras, lo cual significa que es ampliamente usado, pero al mismo tiempo significa que es usado por principiantes.

- SCRATCHPAD: Un sistema muy nuevo, con disponibilidad extremadamente restringida, debido a una estructura completamente diferente, puede llegar a tener las limitaciones de otros sistemas, pero puede ser el prototipo para las siguientes generaciones de los sistemas de álgebra computacional.

Con excepción de SCRATCHPAD, los otros tres sistemas mencionados y muchos otros que no se mencionaron, son muy amigables al usuario, la aceptación de la sintaxis superficial de los lenguajes no es la misma, la librería disponible de funciones varía desde docenas hasta miles, la estructura interna de los sistemas varía considerablemente, pero todas siguen las propiedades siguientes:

- La programación es principalmente interactiva: Los usuarios en teoría no conocen la forma o el tamaño de sus resultados, y deben por lo tanto ser capaces de intervenir en cualquier momento.

- La mayoría de los datos trabajados son sobre expresiones matemáticas, las cuales, al menos en representación externa es de la forma en que estamos acostumbrados a verlas.

- El lenguaje usado es semejante al ALGOL.

- El lenguaje de implementación es con frecuencia LISP; en algunos casos, los datos son listas estructuradas y estructuras de arboles, y el manejo de memoria es dinámico con recuperación automática del espacio disponible.

Accesibilidad de los sistemas de algebra computacional.

Los sistemas de algebra computacional pueden ser usados generalmente, sobre grandes maquinas cuyos sistemas operativos trabajan con memoria virtual. Para trabajar sin gran dificultad, un megabyte de memoria principal parece ser la cantidad más pequeña necesaria.

El sistema más económico es obviamente MUMATH, mientras que el más costoso en memoria de trabajo es SCRATCHPAD, ya que requiere de 8 megabytes de espacio de trabajo. MACSYMA es probablemente el que necesita mayor espacio en disco, ya que el código compilado es de megabytes, y al cual debe ser agregado varios megabytes de líneas de documentación.

MACSYMA corre sólo en VAX, MULTICS. REDUCE, puede ser usado en casi todos los modelos de maquinas grandes o de mini computadoras como: Cyber, IBM series 360, 370, MULTICS, y todos los sistemas con UNIX.

En casi todas las micro computadoras puede ser usado muMATH, mientras que SCRATCHPAD solo trabaja en los sistemas IBM VM/CMS.

Aproximadamente hace treinta años la computadora fue usada por primera vez para realizar calculos algebraicos. Los programas resultantes pudieron diferenciar una gran cantidad de expresiones simples. Sin embargo, tales programas solo fueron accesibles a aquellas grandes computadoras conocidas como macrocomputadoras.

Afortunadamente, la computadora personal esta cambiando esto, ya que pueden correr programas de algebra existentes con impresionante facilidad.

La idea de hacer manipulación algebraica en las computadoras no es nuevo. Desde 1960, numerosos programas habian aparecido en el mercado, con lo cual se intentaba mostrar que el campo científico, puede ir más allá del área puramente matemático, generalmente atribuido a las computadoras.

El lenguaje LISP data de este periodo y abrio el camino para la primera demostración espectacular de las siguientes posibilidades: integración formal y prueba de teoremas.

Pero el problema, es mucho más grande que el manejo o cálculo numérico; ya que nunca ha habido una estandarización. Esta es una de las razones por las que estas posibilidades han sido ignoradas por los científicos y la industria. Por mucho tiempo también, el uso de estos sistemas había sido limitado a cierto tipo de máquina y obviamente esta limitación no ayuda en nada al conocimiento de ellas.

Características de los sistemas de algebra computacional.

La más intuitiva, aunque algunas veces restrictiva característica de los sistemas de algebra computacional, es decir que ellos están hechos para manipular cualquier fórmula científica y de ingeniería.

Una fórmula matemática la cual es descrita en un lenguaje usual (Fortran, Pascal, Basic,...) puede solamente evaluarse numericamente, una vez que las variables y parametros han tomado valores numericos.

En un lenguaje que nos permite manipulaciones algebraicas, la misma formula puede ser evaluada numericamente, pero puede ser objeto también de transformaciones formales como: diferenciación, desarrollo de series, varias expansiones, y aun la integración.

Como regla general, los sistemas de algebra computacional deben seguir dos requerimientos:

- Proporcionar un conjunto de comandos basicos preprogramados, con los cuales indicar a la maquina los calculos que ha de realizar y que procesos correr.
- Ofrecer un lenguaje de programación el cual nos permita definir comandos de alto nivel o procedimientos para enlazarnos con el conjunto de comandos.

Sistema de algebra computacional 'MAT-UNAM'.

Se diseñó para científicos, ingenieros, y otros tipos de profesionales y estudiantes.

Resuelve:

- + Ecuaciones lineales simples de una variable.
- + Ecuaciones no lineales.
- + Polinomios.
- + Derivadas e integrales definidas.

También:

- + Trabaja con desigualdades.
- + Grafica e imprime las gráficas de funciones.
- + Genera reportes.

MAT-UNAM contiene varias construcciones de funciones:

- + La familia de funciones trigonométricas.
- + Las funciones logaritmo y exponencial.
- + una gran cantidad de funciones útiles.

MAT-UNAM contiene un editor de texto, el cual se usa para introducir las ecuaciones y funciones. Este editor es similar al editor de sidekick o turbo pascal.

Requerimientos de hardware.

MAT-UNAM corre en computadoras de la familia IBM PC, incluyendo XT y AT. Además requiere:

- + 640 K de RAM
- + Monitor de 80 columnas.
- + Tarjetas gráficas CGA, EGA, VGA o HERCULES.

MAT-UNAM reconoce los siguientes símbolos:

Operadores aritmeticos:

+ - * / ^ ()

Funciones:

exp, ln, sin, cos, abs

Variables:

Puede ser una secuencia de letras, digitos y periodos, empezando con una letra.

Para entrar en MAT-UNAM sólo se debe escribir MATUNAM y presionar la tecla de enter. Entonces se notara en la pantalla dos características: el menu principal y la línea de estado.

El menu presenta una lista de opciones de las cuales se puede escoger. La línea de estado lista las teclas que realizan una función especial.

La forma de seleccionar elementos del menu es por el método del cursor. Esto es, moviendose de derecha a izquierda o viceversa con las teclas de flecha. Cuando el elemento que nos interesa esta en video inverso, presionar la tecla enter.

MAT-UNAM hace uso del manejo de ventanas. Podría pensarse que una ventana es un lugar donde suceden cosas, cuando se interactua directamente con el programa.

Algunas ventanas muestran un caracter (prompt), requiriendo más información. Otras despliegan los resultados de una operación en particular. Algunas otras, tales como las ventanas de edición, permiten la entrada de datos o textos.

Por otra parte, MAT-UNAM no es sensitivo, esto significa que no reconoce la diferencia entre letras mayúsculas a minúsculas. Por ejemplo, el archivo nombrado como PRUEBA, prueba y Prueba, todos son el mismo para MAT-UNAM.

Lo que debe saberse acerca de la sintaxis de las ecuaciones:

- El operador exponenciación es el símbolo ^, por ejemplo x al cubo se escribe x^3 .
- La multiplicación no se realiza implícitamente.
- La multiplicación se denota con un asterisco (*), por ejemplo tres veces z se escribe $3*z$.
- Para escribir comentarios empezar y terminar con comillas (").

Por ejemplo:

"ecuación de prueba"

Para gráficar una ecuación se debe:

- Incluir al menos una función definida por el usuario

; $f(x,y) = x^2 + 5*y*x$;

o bien:

- Leer un archivo generado con anterioridad por MAT-UNAM.

con = le indicamos que $f(x,y)$ es una función definida por el usuario. podemos escoger la opción de graficar (GRAFICAR), y observaremos la gráfica de esta ecuación.

Puede haber hasta 10 funciones en un archivo. El argumento de una función puede ser una variable, una constante o una función. El tamaño del nombre de las variables y funciones es hasta de 15 caracteres.

Las funciones definidas por el usuario satisfacen dos necesidades: proporciona una manera de definir funciones complicadas, y permite a MAT-UNAM generar gráficas y reportes.

Orden de evaluación de operadores matemáticos:

OPERADOR	ORDEN DE EVALUACION	DIRECCION
()	1 (primero)	izq - der.
^	2	der - izq.
* /	3	izq - der.
+ -	4 (último)	izq - der.

Elección de la alternativa mas viable.

La primera alternativa es poco menos que probable debido a que es muy difícil obtener el código fuente, y en caso de obtenerlo el costo sería elevado, por lo que se descarta.

La segunda alternativa podría tener mayor peso que la primera, sin embargo, de los sistemas de algebra computacional. hasta la fecha ninguno ejecuta modulos externos al paquete. Así que también descartamos esta alternativa.

La tercera alternativa es la más adecuada considerando las limitantes expresadas, por las dos alternativas anteriores, y además que la mayoría de los paquetes de algebra computacional se desarrollaron para equipos denominados main frames. Lo cual es una problemática más, ya que nuestro objetivo es que sea creado para equipos PC.

Selección del lenguaje de programación.

Es importante observar, que para poder decidir que lenguaje de programación se utilizará, hay que considerar las características de estos. En este caso, se han seleccionado tres lenguajes de programación, estos son: pascal, C++, smalltalk.

Dichos lenguajes se encuentran dentro de la clasificación denominada orientada a objetos. A continuación se expondrá un aspecto comparativo entre los lenguajes seleccionados, para poder decidir cual de ellos utilizaremos.

La siguiente tabla muestra el código de programación de cada lenguaje. Considerando características como son : asignación de una variable, un conjunto de expresiones, llamado a funciones con una o más variables, variables con índice (arreglos), comparaciones, ciclos iterativos, regreso de resultados de una función, manejo de memoria (reservado y liberación).

pascal	smalltalk	C++
a:=b+c	a:=b+c	a=b+c
x:=0; y:='algo'; z:=w	x:=0. y:='algo'. z:=w	x=0; y='algo'; z=w;
a:=fun(dato)	a:=dato fun	a=fun(dato)
x:=max(x1,x2); y:=sum(p,q)	x:=x1 max:x2. y:=p+q	x=max(x1,x2); y=sum(p,q);
x:=a[i]; a[i+1]:=y; a[i+1]:=a[i];	x:=a at i. a at:i+1 put:y. a at:i+1 put (a at:i)	x=a[i]; a[i+1]=y; a[i+1]=a[i];
if a < b then a:=a+1;	a < b if true:[a:=a+1]	if(a<b) a++;
while i<10 do begin sum:=sum+a[i]; i:=i+1; end; for i:=1 to 10 do a[i]:=0;	[i<10] while True:[sum:=sum+(a at i). i:=i+1]. 1 to 10 do:[i] a at:i put:0]	while(i<10) { sum+=a[i]; i++; } for (i=1;i<=10;i++) a[i]=0;
hola:=ok; return	^ok	return ok;
new(p); dispose(p);	p:=Array new:5	p=new char[5]; free(p);

Tabla 1. Código generado por cada lenguaje.

Ahora crearemos un tabla donde clasificamos la cantidad de código que genero cada uno de ellos, de acuerdo a una clasificación progresiva de código enumerada como 1, 2 y 3. Donde nivel 1, es el que menos código genero, y el nivel 3, el que más código genera.

	pascal	smalltalk	C++
1.- Asignación de una variable	3	2	1
2.- Serie de expresiones	3	3	1
3.- Función un argumento	3	1	2
4.- Función dos argumentos	3	1	2
5.- Arreglos	2	3	1
6.- Condicionales (if)	2	3	1
7.- Ciclos iterativos	3	2	1
8.- Retorno de una función	3	1	2
9.- Manejo de memoria	3	1	2
T o t a l e s	25	16	13

Como puede observarse el lenguaje que genera menor código es C++. Por otra parte, los tres lenguajes necesitan ser compilados. Así pues, al observar cada uno de ellos en la compilación, se obtuvo lo siguiente: el lenguaje pascal fue el más rápido, debido a que es de una sola pasada y su manejo es directamente en memoria.

Le sigue C++, este es de dos pasadas y hace uso de memoria y acceso a disco. Finalmente smalltalk, el cual también tiene un compilador de dos pasadas, la primera de ellas es muy rápida, pero la segunda es extremadamente lenta.

Después de observar las características anteriores, se ha decidido realizar la programación en el lenguaje C++, por mostrar las mejores características.

CAPITULO III

DISEÑO DEL PAQUETE

El diseño orientado a objetos (DOO), como otras metodologías de diseño orientadas a la información, crea una representación del dominio del problema en el mundo real y lo transforma en un dominio de solución que es un software de aplicación.

A diferencia de otros métodos, el DOO da como resultado un diseño que interconexiona los objetos de datos (elementos de datos) y las operaciones de procesamiento, de forma que modulariza la información y el procesamiento en vez de sólo el procesamiento.

La naturaleza única del diseño orientado a objetos está ligada a su habilidad para construir, basándose en tres conceptos importantes de diseño del software: abstracción, ocultación de la información y modularidad.

Todos los métodos de diseño buscan la creación de software que exhiba estas características fundamentales, pero sólo DOO da un mecanismo que facilita al diseñador adquirir los tres sin complejidad o compromiso.

Surgimientos del Diseño Orientado a Objetos.

Los objetos y las operaciones no son un nuevo concepto de programación, pero sí lo es el diseño orientado a objetos. En los primeros días de la computación, los lenguajes ensambladores

facilitaban a los programadores la utilización de las instrucciones máquina (operadores) para manipular los elementos de datos (operandos).

El nivel de abstracción que se aplicaba al dominio de la solución era muy bajo. Conforme aparecieron los lenguajes de programación de alto nivel (p. ej. FORTRAN, ALGOL, COBOL), los objetos y operaciones del espacio de problemas del mundo real podían ser modelados mediante datos y estructuras de control predefinidas, que estaban disponibles como partes del lenguaje de alto nivel.

En general, el diseño de software se enfocaba sobre la representación del detalle de procedimientos usando el lenguaje de programación elegido. Los conceptos de diseño, tales como refinamientos sucesivos de una función, modularidad y, posteriormente, programación estructurada, fueron introducidos entonces.

Durante los años 1970, se introdujeron conceptos tales como abstracción y ocultación de la información, y emergieron métodos de diseño conducidos por los datos, pero los que desarrollaban software aún se encontraban sobre el proceso y su representación. Al mismo tiempo, los lenguajes de alto nivel modernos (p. ej. Pascal, 'C', etc.) introdujeron una variedad mucho más rica de tipos y estructuras de datos.

Aunque los lenguajes de alto nivel convencionales (lenguajes a partir de FORTRAN y ALGOL) evolucionaron durante los años 1960 y 1970, los investigadores estaban trabajando mucho sobre una nueva clase de lenguaje de simulación y construcción de prototipos, tales como SIMULA y Smalltalk.

En estos lenguajes, la abstracción de datos tenía una gran importancia, y los problemas del mundo real se representaban mediante un conjunto de objetos de datos, a los cuales se les añadía el correspondiente conjunto de operaciones. El uso de estos lenguajes era radicalmente diferente del uso de los lenguajes más convencionales.

El diseño orientado a objetos ha emergido durante los últimos 15 años. Los primeros trabajos en diseño de software pusieron la base estableciendo la importancia de la abstracción, ocultación de la información y de la modularidad en la calidad del software.

En algunos aspectos, el método de diseño orientado a estructura de datos tales como Desarrollo de Sistemas Estructurados de Datos (DSED), puede verse como orientados a objetos.

Durante los años 1980 la rápida evolución de los lenguajes de programación Smalltalk y Ada causaron un creciente interés en DDO. Actualmente el DDO se está usando en aplicaciones de diseño de software que van desde animación, manejadores de bases de datos, paquetes de tutoría hasta telecomunicaciones.

Para conseguir un diseño orientado a objetos, debemos establecer un mecanismo para :

- 1) La representación de la estructura de datos;
- 2) La especificación del procesos, y
- 3) El procedimiento de llamada.

Un objeto es un componente del mundo real que se transforma en el dominio del software. Es decir, un objeto es normalmente un procedimiento o consumidor de información o un elemento de información.

Un objeto esta constituido por una parte publica y otra privada. La parte privada de un objeto es la estructura de datos y el conjunto de operaciones para la estructura de datos.

En el objeto la parte publica que es su interface, los mensajes se mueven a través de la interface y especifican qué operaciones del

objeto se desean, pero no cómo se va a realizar la operación. El objeto que recibe un mensaje determina cómo se implementa la operación solicitada.

A diferencia de otros conceptos de diseño que son independientes del lenguaje de programación, la implementación de las clases y objetos varía del lenguaje de programación usado. Por esta razón, la discusión genérica precedente puede requerir modificación en el contexto de un lenguaje de programación específico.

Una descripción de diseño de un objeto (una instancia de una clase) puede tener una de las dos formas siguientes:

1.- Una descripción de protocolo que establece la interface de un objeto, definiendo cada mensaje que puede recibir el objeto y la operación correspondiente que el objeto ejecuta cuando recibe el mensaje, y

2.- Una descripción de la implementación que muestra los detalles de implementación para cada operación implicada en un mensaje que se pase al objeto. Los detalles de implementación incluyen información sobre la parte privada del objeto, esto es, los detalles internos sobre la estructura de los datos y los detalles de los procedimientos que describen las operaciones.

Una descripción de la implementación consta de la siguiente información:

- 1) Una especificación del nombre del objeto y referencia a la clase;
- 2) Una especificación de la estructura de datos privada con una indicación de los elementos y tipos de datos, y
- 3) Una descripción de procedimientos de cada operación o, alternativamente, apuntadores a tales descripciones de procedimientos.

En la etapa actual de evolución, la metodología DDD combina elementos de las tres categorías de diseño: diseño de datos, diseño de arquitectura y diseño de procedimientos.

Para la identificación de los objetos, se crean abstracciones de datos. Definiendo operaciones, se especifican los módulos y se establece una estructura para el software. Desarrollando un mecanismo para usar los objetos se describen las interfaces.

Pasos para el diseño Orientado a Objetos.

- 1.- Definir el problema.
- 2.- Desarrollar una estrategia informal para la realización del software en el dominio del problema en el mundo real.

3.-Formalizar la estrategia usando los siguientes subpasos:

- a) Identificar los objetos y sus atributos.
- b) Identificar las operaciones que pueden aplicarse a los objetos.
- c) Establecer interfaces para mostrar las relaciones entre los objetos y las operaciones.
- d) Decidir los aspectos del diseño detallado que harán una descripción de la implementación para los objetos.

4.- Repetir los pasos 2,3 y 4 recursivamente hasta que se cree un diseño completo.

Debe observarse que los primeros dos pasos se ejecutan realmente durante el análisis de requerimientos del software.

El método de diseño orientado a objetos presentado está orientado hacia el desarrollo del software en lenguajes de programación tales como Ada. El método no se enfoca explícitamente hacia varios conceptos importantes orientados al objeto (p. ej. herencia, mensajes) que pueden hacer al DOD incluso más poderoso.

A continuación se describe un método alternativo para el DOD que se ha obtenido a partir del desarrollo de software en lenguajes de programación tales como Smalltalk, C++; lenguajes que soportan directamente la abstracción, herencia, mensajes y todos los otros conceptos del DOD.

Pasos del diseño.

El método DDD es apropiado para el diseño preliminar. El objetivo primario es definir y caracterizar las abstracciones, de forma que se obtenga una definición de todos los objetos importantes, métodos (operaciones) y mensajes.

1.- Identificar las clases de datos para cada subsistema.

Trabajando a partir del documento de requerimientos, el proceso de clasificación debe ejecutarse de la forma más descendente posible, aunque muchas veces las clases se mencionan explícitamente en los requerimientos. Frecuentemente, las clases corresponden a objetos físicos dentro del sistema que se esté modelando.

Si éste no es el caso, es útil hacer uso de analogías, obtenidas de la experiencia del diseñador en el diseño de otros sistemas. Esto es, por ahora, el paso más difícil en el proceso de diseño y la selección de estas clases influye en la arquitectura global del sistema.

2.- Identificar los atributos de cada clase.

Los atributos se convierten en variable de los métodos para manipular los datos para cada clase. Muchas veces, si las clases corresponden a objetos físicos, son obvias las variables requeridas para los métodos.

3.- Identificar las operaciones de cada clase.

Las operaciones son los métodos (o procedimientos) de cada clase. Algunos métodos acceden y actualizan variables de clase, mientras que otros ejecutan operaciones singulares de la clase. No especificar los detalles de la implementación del método ahora, sólo las funcionalidades.

Si la nueva clase hereda de otra clase, inspeccionar los métodos de esa clase para ver si necesita invalidar alguna por la nueva clase. Dejar el diseño interno de los métodos hasta la etapa del diseño detallado, en donde puede usarse una técnica más convencional de diseño.

4.- Identificar la comunicación entre los objetos.

Este paso define el enlace que hay entre unos objetos y otros. Aquí, definir una correspondencia entre los métodos y la manera en que se llaman a los métodos. Incluso si no se ha planificado una implementación orientada a objetos, el equipo de diseño decide sobre este protocolo de comunicación entre objetos.

5.- Probar el diseño.

Las pruebas consisten en hacer llamados a los objetos, y así probar la habilidad del diseño para cumplir los requerimientos del sistema. Cada función debe satisfacer, a nivel de usuario, la especificación de requerimientos.

6.- Aplicar la herencia donde sea apropiada.

Si el proceso de clasificación de datos en el paso 1 se ejecuta de forma descendente, introducir allí la herencia. Sin embargo, si las clases se crean en forma ascendente (frecuentemente debido a que los requerimientos denominan directamente a las clases), aplicar aquí la herencia, antes de ir a otro nivel de clasificación.

El objetivo es reutilizar tanto como se pueda los datos y/o los métodos que hayan sido diseñados. En este paso, emergen frecuentemente datos comunes y operaciones, y estas variables comunes de la clase y métodos, pueden combinarse en una nueva clase. Esta clase puede tener o no significado como objeto por sí mismo. Si su único propósito es coleccionar variables y métodos de clase comunes, se llama una clase abstracta.

El diseñador repite estos pasos en cada nivel de clasificación. A través de sucesivos refinamientos del diseño, la visión del diseñador del sistema cambia, dependiendo de las necesidades de cada momento. Cada nivel de clasificación se implementa en un nivel inferior, hasta que se alcance un punto en el que la clasificación corresponda con un elemento primitivo del diseño.

Un vistazo a smalltalk y C++.

Las aplicaciones de smalltalk pueden verse en las áreas de simulación, sistema expertos, tutores, base de datos, etc. El lenguaje smalltalk es considerado como el lenguaje de programación orientado a objetos puro. De tal forma que tiene un aprovechamiento sobre la abstracción de datos.

Esto significa que el software es altamente reusable, el código generado y el uso de éste, es un estilo prototipo de desarrollo de software. Algunas características adicionales son:

- El componente más importante en un sistema de computación es el usuario.
- La programación debe ser una extensión natural del pensamiento.
- La programación debe ser un proceso dinámico y evolutivo con el modelo de aprendizaje humano.

El lenguaje C++, llamado originalmente "C con clases", fue desarrollado por Bjarne Stroustrup en los laboratorios Bell, en New Jersey en 1983.

Mientras el lenguaje C es excelente para proyectos de pequeño y mediano tamaño, es restringido para proyectos extremadamente grandes. El lenguaje C++ fue inventado para permitir un manejo más sencillo de proyectos grandes.

¿ Que es la programación orientada a objetos ?

La programación orientada a objetos es una nueva forma de aprovechar el trabajo de programación. El aprovechamiento de la programación ha cambiado dramáticamente desde la invención de la computadora. La principal razón para este cambio es el acondicionar el incremento de la complejidad de los programas.

Por ejemplo, cuando las computadoras fueron inventadas, la programación era en forma binaria usando instrucciones del panel. Como el tamaño de los programas era de unos cuantos cientos de instrucciones este tipo de aprovechamiento funcionaba.

Como los programas crecieron, el lenguaje ensamblador fue inventado, así que los programas podrían ser más grandes, incrementándose la complejidad de los mismos, por el uso de representaciones simbólicas de las instrucciones de máquina.

Como los programas continuaron creciendo, los lenguajes de alto nivel fueron introducidos, para proporcionar al programador más

herramientas con las cuales manejar esta complejidad, el primero de ellos fue el lenguaje FORTRAN.

En los sesentas surge la programación *estructurada*, este es el método en los que se encuentran lenguajes como "C", y PASCAL.

Usando la programación estructurada, fue posible por primera vez escribir programas moderadamente complejos de una manera sencilla.

Sin embargo, aún con los métodos de la programación estructurada, los proyectos de investigación de cierto tamaño eran incontrolables.

La razón, es que la complejidad excedía las técnicas de la programación estructurada que el programador podía manejar.

Actualmente, muchos proyectos estan cerca o en el punto donde la programación estructurada no puede ser aprovechada por la complejidad de los mismos. Para solucionar este problema, fue inventada la programación orientada a objetos.

La programación orientada a objetos toma las ideas de la programación estructurada y las combina con gran poderio. La programación orientada a objetos, permite de una manera sencilla descomponer un problema en subgrupos que realcionan las partes del problema. Entonces, pueden trasladarse estos grupos en unidades autocontenidas llamadas *objetos*.

Todos los lenguajes de programación orientados a objetos tienen tres cosas en común: objetos, polimorfismo y herencia.

Objetos.

La más importante característica de un lenguaje orientado a objetos es el *objeto*. Un objeto es una entidad lógica que contiene datos y código que manipula esos datos. Dentro de un objeto, código y/o datos pueden ser privados del objeto, y son inaccesibles a objetos externos. En este sentido un objeto, proporciona un nivel significativo de protección.

El ligado de código y datos de esta manera es llamado *encapsulación*. Un objeto es una variable del tipo definido por el usuario. Puede parecer extraño a primera instancia que un objeto, el cual liga tanto código y datos, sea una variable.

Sin embargo, en la programación orientada a objetos, este es precisamente el caso, cuando se define un objeto, implícitamente se esta creando un nuevo tipo de dato.

Polimorfismo.

La programación orientada a objetos soporta el polimorfismo, lo cual esencialmente significa que un nombre puede ser usado para varias relaciones, pero para propósitos ligeramente diferentes. El propósito del polimorfismo es permitir que un nombre pueda ser usado para especificar una clase general de acciones.

Sin embargo, dependiendo del tipo de dato que se esta ocupando, una instancia especifica del caso general es ejecutada. Por ejemplo, si tuvieramos tres tipos de pilas, de tal forma que una pila usará valores enteros, otra de punto flotante y otra de tipo long. si se crearán las funciones push() y pop(), el compilador seleccionaría la rutina correcta, dependiendo del tipo de datos de la función de llamada.

Los primeros lenguajes orientados a objetos fueron interpretes, asi que el polimorfismo fue por supuesto soportado en el tiempo de corrida. Sin embargo, C++ es un lenguaje compilado, por lo tanto el polimorfismo es soportado tanto en el tiempo de corrida como en el de compilación.

Herencia.

La herencia es el proceso por el cual un objeto puede adquirir las propiedades de otro objeto. Esta es importante porque soporta el concepto de clasificación. Por ejemplo, una manzana roja es parte de la clasificación manzana, la cual a su vez es parte de la clase fruta, que esta bajo la clase alimento.

Sin el uso de las clasificaciones, cada objeto tendría que definir explicitamente todas sus características. Sin embargo, usando clasificaciones, un objeto necesita definir sólo aquellas cualidades que la hacen única dentro de su clase. Puede además, heredar aquellas cualidades que comparte con las clases más generales.

Clases y objetos.

En C++, una de las más importantes características es la clase (*class*). Para crear un objeto, debe definirse en forma general con la palabra reservada *class*. Una clase es similar a una estructura, y puede contener partes tanto públicas como privadas. Por default, todos los elementos de la clase son privados.

Tener partes privadas significa que estas no pueden ser accedidas por cualquier función que no sea miembro de la clase. Esta es una manera de lograr la encapsulación.

Para hacer partes públicas, debe declararse antes de ellas la palabra reservada *public*, de tal manera que todas las variables o funciones definidas como públicas son accesibles para cualquiera de las otras funciones en el programa.

Constructores y destructores.

Es muy común que alguna parte de un objeto requiera inicializarse antes de ser usada. C++, permite a los objetos inicializarse ellos mismos cuando son creados. Esta inicialización automática es realizada a través del uso de la función *constructor*. Una función constructor es una función especial que es miembro de la clase y tiene el mismo nombre de esa clase.

El complemento del constructor es el destructor. En diversas circunstancias, un objeto necesitará realizar alguna acción o acciones cuando este sea destruido. Por ejemplo, un objeto necesitará liberar memoria que previamente tomó. El destructor tiene el mismo nombre que el constructor pero está precedido por ~.

Diseño de los módulos del sistema MAT-UNAM.

El sistema estará constituido por 4 módulos principales, que son :

- Editor.
- Evaluar.
- Graficar.
- Tutor.

A continuación se describe a detalle, la forma en que se diseñarán cada uno de ellos.

Módulo Editor.

Para este módulo se han observado las características de los editores comerciales, y se han seleccionado aquellas, que se consideran las más idóneas para la aplicación que tendrá el sistema MAT-UNAM.

Dentro de las características que han sido seleccionadas se tienen las siguientes:

- Pantalla dividida en tres zonas
 - a) Menus.
 - b) Area de trabajo.
 - c) Línea de estado.

- Facilidad de movimiento en la pantalla en las cuatro direcciones: arriba, abajo, izquierda, y derecha.

- Borrado de:
 - a) Caracter a la izquierda.
 - b) Caracter bajo el cursor.
 - c) Línea.

- Cambio de inserción y sobreescritura, según se desee.

- Salvado de archivos en modo texto.

- Carga de archivos de modo texto.

- Muestra de directorio de la ruta actual.

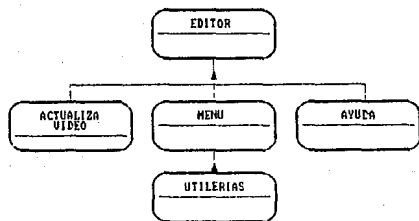
- Cambio de directorio y/o unidad.

- Ayuda del paquete.

- Salida del sistema en forma convencional.

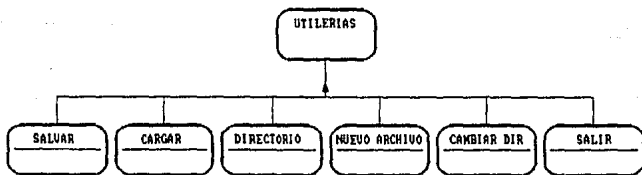
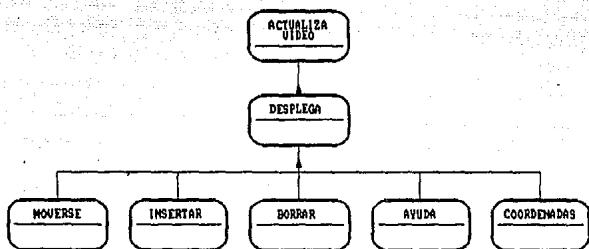
Por otra parte, el manejo de los menus permitirá al usuario seleccionar la opción que desee, de una manera sencilla. Logrando un diálogo con el usuario aún cuando se trate de pequeños detalles.

A continuación se muestra el diagrama a bloques de dicho modulo.



Modulo Editor.

A partir del diagrama anterior, se pueden obtener características más detalladas de este módulo, como se ve en las siguientes figuras:



Modulo Evaluar.

Para poder realizar el diseño de dicho modulo. se considero pertinente, analizar primero las características de un compilador. Así pues, entre ellas tenemos: tipos de compiladores, partes funcionales, accesibilidad de programación, eficiencia, etc.

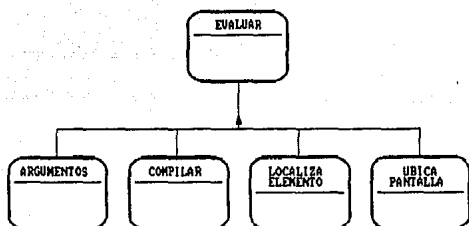
Por lo tanto, y teniendo presentes las características mencionadas, se decidió realizar un compilador con las siguientes características:

- + Scanner- Por medio de tablas.
- + Parser- Dirigido por sintaxis.
- + Código intermedio.

Esta decisión se tomo, ya que sólo tendremos dos pequeñas tablas en el scanner. Además, el parser por ser dirigido por sintaxis no ocupará espacio adicional de memoria, como en el caso de un parser con manejo de tablas.

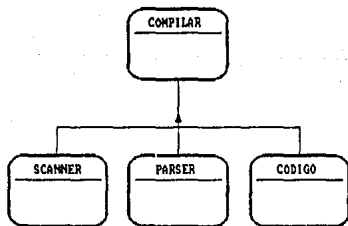
El código intermedio será manejado a nivel de memoria, de tal manera que se intenta hacer el mínimo de accesos a disco. Por lo que, la ejecución será más rápida y la memoria será liberada.

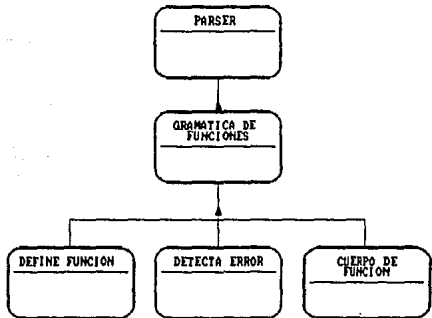
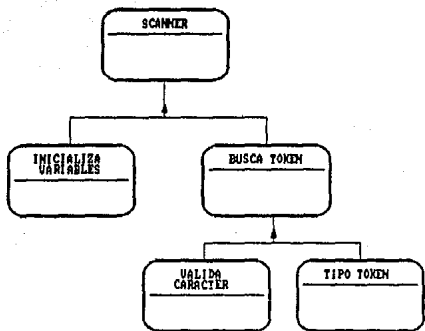
Finalmente se obtuvo un modelo más específico de este modulo como se ve en la siguiente figura:

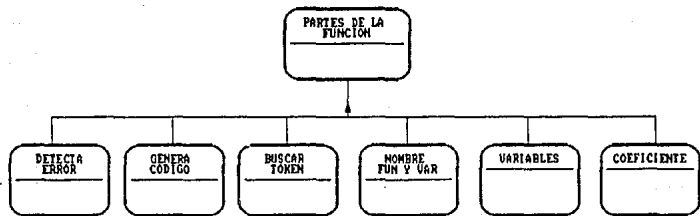
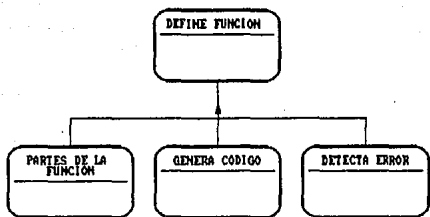


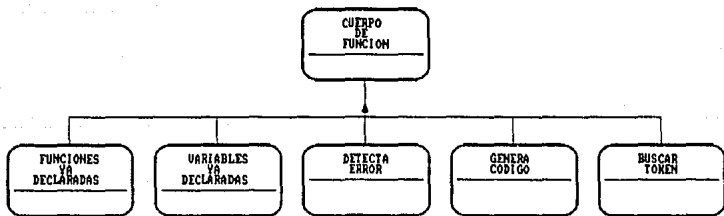
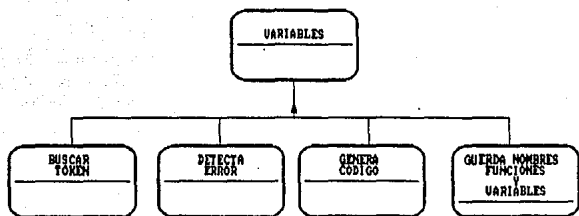
Modulo Evaluar.

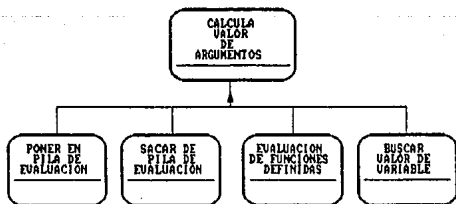
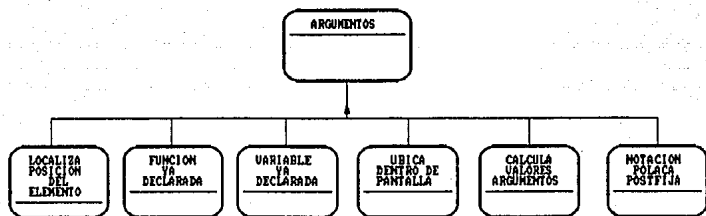
De igual manera, que en el modulo editor, se han obtenido aspectos más específicos de este modulo, como se observa en las figuras siguientes.

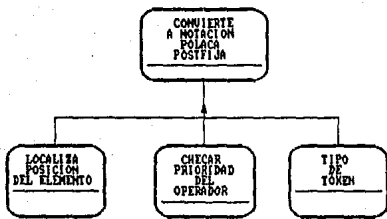








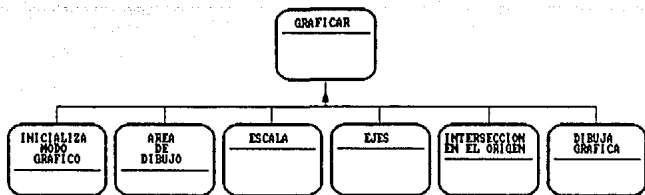




Modulo Graficar.

El presente modulo se encarga de graficar las funciones que serán definidas en el editor. De los aspectos más importantes que se deben contemplar, se encuentran los diferentes tipos de tarjetas gráficas. Por lo que, se deberá crear un procedimiento generalizado, sin importar de que tarjeta gráfica se trate.

Por lo anterior, se decidió modelarlo como sigue:



Modulo Graficar.

Inicializa modo gráfico : Submodulo que se encargará de checar el tipo de video gráfico e inicializarlo.

Area de dibujo : Submodulo que se encarga de seleccionar el área más adecuada de dibujo, dependiendo del tipo de tarjeta gráfica.

Escala : Submodulo que obtendrá la escala adecuada de dibujo, dependiendo del tipo de tarjeta gráfica.

Ejes : Submodulo que dibujará y escalará los ejes X y Y.

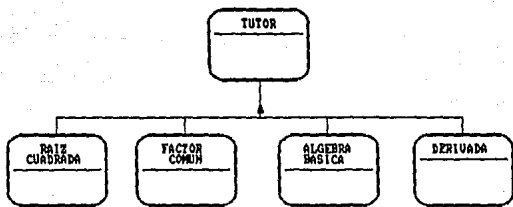
Intercepción en el origen : Submodulo que calculará la posición en la cual los ejes se interceptan. Para que la presentación gráfica sea proporcional a los ejes.

Dibuja gráfica : Submodulo que unirá los puntos y mostrará la gráfica, ya sea en pantalla o en pantalla e impresora.

Modulo Tutor.

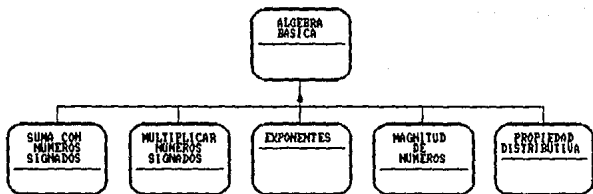
Este modulo es muy importante, pero también es importante, definir los alcances y limitaciones de lo que se pretende. El área de las matemáticas es muy amplio y complejo, por lo que limitar esta área también lo es. Es por ello sólo nos avocaremos hacia el área del algebra.

Sin embargo, consideraremos un caso simple de cálculo diferencial, así que, el modelo que se pretende es el siguiente:



Módulo Tutor.

La desición de dicho modelo se basa en que, la mayor parte de los estudiantes de matemáticas tienen deficiencias en algebra, pero es más acentuada en algebra básica. De esta última, se obtienen las características que se ven en la siguiente figura:



Hasta el momento, se han analizado, delimitado y diseñado, cada uno de los módulos. El paso siguiente será programar cada uno de ellos.

CAPITULO IV
PROGRAMACION Y PRUEBAS

Programación.

En los sistemas catalogados como buenos bajo el punto de vista de ingeniería, existe una tendencia natural a usar los recursos críticos de forma eficiente.

La eficiencia del código fuente está directamente unida a la eficiencia de los algoritmos definidos durante el diseño detallado. Sin embargo, el estilo de codificación puede afectar la velocidad de ejecución y a los requerimientos de memoria.

Muchos compiladores incluyen opciones de optimización que generan automáticamente código eficiente, al evaluar los ciclos, usando aritmética rápida, generando código reentrante, y aplicando algoritmos relacionados con la eficiencia.

Este es el caso de C++, el lenguaje de programación que se ha seleccionado para la programación del sistema MAT-UNAM. A continuación se presenta el código fuente de los objetos scanner, parser y editor del sistema MAT-UNAM.

```
=====
Aquí se hace la declaración de los objetos scanner y
parser.
===== */
```

```
class scanner {
    typedef struct {char l[80];} largo;
    char ch;
    largo *tokens, *limite;
    int edo_act, salida, auxiliar;
    int lim, inicio, limi;
public:
    char tok[20];
    int pila, con, lin;
    int inicia(largo *iniarch, largo *finarch);
    int quitacomen(void);
    void checacar(char c);
    int bustoken(void);
    int tipotoken(int b, char *t1);
};
```

```
/*=====
El objeto parser hereda todas las funciones que realiza
el objeto scanner.
===== */
```

```
class parser:public scanner {
    typedef struct {char p[20]; int dir;} guarda;
    typedef struct {char p[20]; double dir;} guarda2;
    typedef struct {char p[20];} guarda1;
protected:
    int ntok, ser, ant, cuenta, ban, vari;
public:
    int mom;
    guarda buscar[16], codi;
    guarda2 var[5];
    guarda1 *probar, *probar1;
    int error(int a);
    int termino(void);
    int coeficiente(void);
    int variables(void);
    int funcion(void);
    int define(void);
    int gramafun(void);
    int guardadas(int r);
    int codigo(void);
    int funya(char *existe);
    int varya(char *existe);
};
```

```
/*=====
Declaración del objeto editor
=====*/
```

```
class editor {
    int i,e,yf,x,y,cl,insert;
    largo *inicio, *fin, *k, *mover;
    char linea[80],blaf[80];
    char nom[80];
public:
    void desplega(void);
    void pre(void);
    void ruido(void);
    void coor(int r);
    void cooc(int c);
    int salvar(void);
    int cargar(void);
    void lim(void);
    void mueve(void);
    void muevek(void);
    void mueve20(void);
    void alicar(void);
    int menu(void);
    int selmenu(void);
    void vid(int t);
    int finlinea(int h, int v, int vi);
    void siglinea(char *plinea, int w);
    int leenom(int x1, int y1);
};
```

Objetivo : Compilación y generación de código intermedio de los archivos de entrada.

```
/*=====
Aquí se definen la tabla de estados y de salida, así
como también, las palabras reservadas y el número de
token de las mismas.
===== */
```

```
typedef struct {char id[6]; int numtoken;}nodo;
```

```
nodo reserv[20] = { "SEN",8,"COS",9,"TAN",10,"COT",11,"CSC",12,
"EXP",13,"LN",14,"ABS",22,"DERIV",23,"FACT",24,
"INTEG",25,"FRAC",26,"ENT",27,"LOG",28,"POLI",29,
"RAIZ",30,"SUMA",31,"POLAR",32,"CART",33,"ATAN",34
```



```

/*
A O E E
: : C R O
Z 9 S ( ) = R R . , F : "
int estados[15][13]={ 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14,
1, 1, 0, 0, 0, 0, 0, 8, 0, 0, 12, 0, 0,
8, 2, 0, 0, 0, 0, 0, 8, 9, 0, 12, 0, 0,
0, 0, 0, 0, 0, 0, 0, 8, 0, 0, 12, 0, 0,
0, 0, 0, 0, 0, 0, 0, 8, 0, 0, 12, 0, 0,
0, 0, 0, 0, 0, 0, 0, 8, 8, 0, 12, 0, 0,
0, 0, 0, 0, 0, 0, 0, 8, 0, 0, 12, 0, 0,
0, 0, 0, 0, 0, 0, 0, 8, 0, 0, 12, 0, 0,
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14,
8, 9, 0, 0, 0, 0, 0, 8, 0, 0, 12, 0, 0,
0, 0, 0, 0, 0, 0, 0, 8, 0, 0, 12, 0, 0,
0, 0, 0, 0, 0, 0, 0, 8, 0, 0, 12, 0, 0,
12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12,
0, 0, 0, 0, 0, 0, 0, 8, 0, 0, 12, 0, 0,
0, 0, 0, 0, 0, 0, 0, 8, 0, 0, 12, 0, 0};

```

```

/*
A O E E
: : C R O
Z 9 S ( ) = R R . , F : "
int edo_sal[15][13]={ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
0, 0, 2, 2, 2, 2, 2, 2, 0, 2, 11, 2, 2,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 11, 3, 3,
4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 11, 4, 4,
5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 11, 5, 5,
6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 11, 6, 6,
7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 11, 7, 7,
8, 8, 8, 8, 8, 8, 8, 8, 0, 8, 8, 11, 8, 8,
0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 11, 2, 2,
9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 11, 9, 9,
10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 11, 10, 10,
11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11,
12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 11, 12, 12,
13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 11, 13, 13};

```

```

/*+++++
AGUI INICIA LA IMPLEMENTACION DEL OBJETO SCANNER
+++++*/

```

```

/*=====
En este procedimiento se inicializan todas las variables
que utiliza el objeto scanner.
===== */

int scanner::inicia(largo *iniarch, largo *finarch)
{
    tokens=iniarch;
    limite=finarch;
    edo_act=salida=0;
    lim1=0;auxiliar=0;
    con=inicio=pila=0;
    return(auxiliar);
};

/*=====
Este procedimiento elimina los comentarios del programa.
===== */

int scanner::quitacomen(void)
{
    while((ch=tokens->l[lim1++]!='"') {
        if(ch==0) {
            lint++;lim1=0;tokens++;
        }
        if(tokens>limite) return(1);
    }
    while((ch=tokens->l[lim1++]==' '));
    checacar(ch=toupper(ch));
    if(ch=="") quitacomen();
    return(0);
}

/*=====
En este procedimiento se chequea que las entradas sean
validas. Ademas indica el tipo de entrada.
===== */

void scanner::checacar(char c)
{
    if (isalpha(c))
        auxiliar=0;
    else
        if (isdigit(c))
            auxiliar=1;
        else
            switch (c) {
                case '+' : auxiliar=2;break;
                case '-' : auxiliar=2;break;
            }
}

```

```

case '/' : auxiliar=2;break;
case '*' : auxiliar=2;break;
case '^' : auxiliar=2;break;
case '(' : auxiliar=3;break;
case ')' : auxiliar=4;break;
case '=' : auxiliar=5;break;
case '\0' : auxiliar=6;lin++;lim1=0;tokens++;
           while ((ch=tokens->l[lim1++])==' ');
           checacar(ch=toupper(ch));break;
case '.' : auxiliar=8;break;
case ',' : auxiliar=9;break;
case ';' : auxiliar=11;break;
case ':' : auxiliar=12;break;
default: auxiliar=7;
};
};

/*=====
   En este procedimiento se obtiene el token.
   =====*/

int scanner::bustoken(void)
{
int i=0;
static char ultimo;
salida=0;
tok[0]='\0';
if (inicio!=0) {
ch=ultimo;
if(ch=="") if(quitacomen()==1) return(40);
checacar(ch);
salida=edo_sal[edo_act][auxiliar];
edo_act=estados[edo_act][auxiliar];
tok[i++]=ch;
inicio=0;
}
while(auxiliar!=7 && salida==0 && tokens<=limite){
while ((ch=tokens->l[lim1++])==' ');
checacar(ch=toupper(ch));
salida=edo_sal[edo_act][auxiliar];
edo_act=estados[edo_act][auxiliar];
tok[i++]=ch;
inicio++;
};
tok[i]='\0';
if(salida==13) {
auxiliar=edo_act=salida=0;
if(quitacomen()==1) return(40);
i=inicio=0;
}
}

```

```

salida=edo_sal[edo_act][auxiliar];
edo_act=estados[edo_act][auxiliar];
tok[i++]=ch;
while(auxiliar!=7 && salida==0 && tokens<=limite){
  while ((ch=tokens->l[limi++]==' ');
  checar(ch=toupper(ch));
  salida=edo_sal[edo_act][auxiliar];
  edo_act=estados[edo_act][auxiliar];
  tok[i++]=ch;
  inicio++;
};
tok[i]='\0';
}
ultimo=ch;
if ((lim=strlen(tok))!=1) tok[--lim]='\0'; else tok[lim]='\0';
return(tipotoken(salida,tok));
};

```

```

/*=====
En este procedimiento se obtiene el tipo y el numero
de token.
===== */

```

```

int scanner::tipotoken(int b, char *t1)
{
  int tipo,numres;
  switch (b){
    case 1 : numres=0;tipo=1;
      while(numres<20 ){
        if (!strcmp(reserv[numres].id,t1))
          { tipo=reserv[numres].numtoken; break; }
          numres++;
      };
      break;
    case 2 : tipo=2;break;
    case 3 : switch (*tok) {
      case '+':tipo=3;break;
      case '-':tipo=4;break;
      case '*':tipo=5;break;
      case '/':tipo=6;break;
      case '^':tipo=7;break;
      default:tipo=0;
      };
      break;
    case 4 : tipo=15;break;
    case 5 : tipo=16;break;
    case 6 : tipo=17;break;
    case 7 : tipo=18;break;
    case 9 : tipo=20;break;
  }
}

```

```

    case 10 : tipo=21;break;
    case 12 : tipo=19;break;
    case 13 : tipo=40;break;
    default : tipo=0;
};
return(tipo);
};

/*****
  AQUI TERMINA LA IMPLEMENTACION DEL OBJETO SCANNER
  *****/

/*****
  AQUI INICIA LA IMPLEMENTACION DEL OBJETO PARSER
  *****/

/*****
  En este procedimiento se despliegan los errores que han
  sido detectados.
  *****/
int parser::error(int a)
{
  typedef struct {char s[40];}errores;
  errores terror[19]={"Se espera nombre de funcion",
                    "Se espera parentesis izquierdo",
                    "Se espera variable",
                    "Se espera parentesis derecho",
                    "Se espera signo igual",
                    "Se espera caracter valido",
                    "Se espera numero o variable",
                    "Cerrando parentesis sin abrirlo",
                    "Funto <.> extra en la funcion",
                    "Uso inadecuado de la coma <,> ",
                    "Se espera punto y coma <;> ",
                    "Signo igual no valido en esta zona",
                    "Funcion no valida en esta zona",
                    "Nombre de funcion declarada 2 veces",
                    "Variable declarada 2 veces",
                    "Operador invalido al inicio de funcion",
                    "Variable o funcion no declarada",
                    "Disco lleno o protegido",
                    "Abriendo comentario sin cerrarlo"};
  gotoxy(1,1);printf("E R R O R linea %2d << %s >>\n",lin,terror[a].s);
  return(-1);
};

```

```

/*****
En este procedimiento se realiza la mayor carga de
trabajo del parser. Este es recursivo, verifica la
correcta sintaxis, en caso de existir un error llama
a la funcion de "error".
*****/

```

```

int parser::termino(void)
{
switch (ntok) {
case 0 : return(ntok=error(5));
case 1 : if(funya(tok)==buscar[0].dir && varya(tok)==var[0].dir)
return(ntok=error(16));
codigo();ant=ntok;break;
case 2 : codigo();ant=ntok;break;
case 3 :
case 4 :
case 5 :
case 6 :
case 7 : codigo();ant=ntok;
if ((ntok=bustoken())!=15){
codigo(); ant=ntok;
ntok=bustoken();
termino();
if (ntok!=16) return(ntok=error(3));
}
else
switch (ntok) {
case 1: case 2:codigo();break;
case 8 : case 9 : case 10: case 11: case 12: case 13:
case 14: case 22: case 26: case 27: case 28: case 30:
case 34:codigo();
if ((ntok=bustoken())!=15) return(ntok=error(1));
if (ban!=0) codigo();ant=ntok;
ntok=bustoken();
if(termino()==-1) return(-1);
if (ntok!=16) return(ntok=error(3));strcpy(tok,"!");codigo();
break;
case 23: case 24: case 25: case 29: case 31: case 32:
case 33:if(ban==1) return(ntok=error(12));
codigo();
if ((ntok=bustoken())!=15) return(ntok=error(1));
if (ban!=0) codigo();ant=ntok;
ntok=bustoken();
if(termino()==-1) return(-1);
if (ntok!=16) return(ntok=error(3));strcpy(tok,"!");codigo();
break;
default:return(ntok=error(6));
};
break;
}
}

```

```

case 8 : case 9 : case 10: case 11: case 12: case 13: case 14:
case 22: case 26: case 27: case 28: case 30:
case 34:codigo():ant=ntok;
if ((ntok=bustoken())!=15) return(ntok=error(1));
if (ban!=0) codigo():ant=ntok;
ntok=bustoken();if(termino()==-1) return(-1);
if (ntok!=16) return(ntok=error(3));strcpy(tok,"");codigo();
break;
case 23: case 24: case 25: case 29: case 31: case 32:
case 33:if(ban==1) return(ntok=error(12));
codigo():ant=ntok;
if ((ntok=bustoken())!=15) return(ntok=error(1));
if (ban!=0) codigo():ant=ntok;
ntok=bustoken();if(termino()==-1) return(-1);
if (ntok!=16) return(ntok=error(3));strcpy(tok,"");codigo();
break;
case 15:if (ant==1 && ban==0) return(ntok=error(12)); codigo():ant=ntok;
ntok=bustoken();
termino();
if (ntok!=16) return(ntok=error(3));
break;
case 20:return(ntok=error(8));
case 17:return(ntok=error(11));
case 21:ant=ntok;break;
case 40:return(ntok=error(18));
};
if (ntok==16 && ser==0) {
ser++;
if(con==1) return(ntok=error(7));
if (ant==15) return(ntok=error(6)); else
if (ban!=0) codigo():ant=ntok;
}
else{
ser=0;
switch (ntok=bustoken()) {
case 1: case 2: case 3: case 4: case 5: case 6: case 7: case 8: case 9:
case 10: case 11: case 12: case 13: case 14: case 15: case 16: case 17:
case 20: case 21: case 22: case 23: case 24: case 25: case 26: case 27:
case 28: case 29: case 30: case 31: case 32: case 33: case 34:
case 40: termino();
break;
};
};
return(ntok);
};

```

```

/*=====
Aquí se checa si existe un signo al principio de la
funcion.
===== */

int parser::coeficiente(void)
{
switch (ntok) {
case 3: case 4: codigo();ant=ntok; ntok=bustoken();break;
case 5: case 6: case 7: return(ntok=error(15));
}
return(termino());
};

/*=====
Aquí se verifica que las variables esten correctamente
declaradas.
===== */

int parser::variables(void)
{
if ((ntok=bustoken())!=1) return(ntok=error(2));ant=ntok;
if(guardadas(1)!=0) return(ntok=error(14));codigo();
if ((ntok=bustoken())==21) variables();
return (ntok);
};

/*=====
Aquí se verifica que la funcion de entrada sea
declarada correctamente.
===== */

int parser::funcion(void)
{
ban=1;
if ((ntok=bustoken())!=1) return(ntok=error(0));ant=ntok;
if (guardadas(0)!=0) return(ntok=error(13)); codigo();
if ((ntok=bustoken())!=15) return(ntok=error(1));ant=ntok;
if (variables() == -1) return(-1); strcpy(tok,"");codigo();
if (ntok!=16) return(ntok=error(3));ant=ntok;
if ((ntok=bustoken())!=17) return(ntok=error(4));ant=ntok;
if ((ntok=bustoken())!=18) return(ntok=error(6));ant=ntok;
if (coeficiente() == -1) return(-1);
ban=0;return(ntok);
};

```



```

/*=====
Este procedimiento verifica si la funcion o variable ya
se encuentra definida. Si aun no lo esta, la define.
===== */

int parser::guardadas(int r)
{
register int i=1;
if (r==0) {
while( i<cuanta) { if (strcmp(buscar[i++].p,tok)==0) return(1); }
strcpy(buscar[cuanta].p,tok);
buscar[cuanta++].dir=mem;
buscar[0].dir=cuanta;
}
else {
while( i<vari) {
if (strcmp(var[i].p,tok)==0 && var[i].dir==cuanta) return(1); i++;
}
i=1;while( i<vari) { if (strcmp(var[i].p,tok)==0) return(0); i++; }
strcpy(var[vari].p,tok);
var[vari].dir=cuanta;vari++;
var[0].dir=vari;
}
return(0);
}

/*=====
Este procedimiento regresa la posicion de la funcion, de
no estar definida, regresara el numero de funciones que
están definidas.
===== */

int parser::funya(char *existe)
{ register int i=1;
while( i<buscar[0].dir) { if (!strcmp(buscar[i].p,existe)) break; i++; }
return(i);
}

/*=====
Este procedimiento regresa la posicion de la variable, de
no estar definida, regresara el numero de variables que
están definidas.
===== */

int parser::varya(char *existe)
{ register int i=1;
while( i<var[0].dir) { if (!strcmp(var[i].p,existe)) break; i++; }
return(i);
}

```

```

/*=====
   En este procedimiento se definen las funciones de usuario.
   ===== */

int parser::define(void)
{
    if (funcion()==-1) return(-1); if (ntok!=19) return(error(10));
    strcpy(tok,"#");codigo();strcpy(tok,"$");codigo();
    if ((ntok=bustoken())==19) define();
    return(ntok);
}

/*=====
   Este procedimiento es el que se encarga de coordinar todas
   las operaciones que se realizan en el parser.
   ===== */

int parser::gramafun(void)
{
    ban=ser=ant=0; mem=lin=1; vari=cuenta=1;
    probar1=probar;

    strcpy(buscar[0].p,"Funciones");
    strcpy(var[0].p,"variables");
    if ((ntok=bustoken())==19) if(define()==-1) return(-1);
    cuenta=mem;
    if(termino()==-1) return(ntok);
    strcpy(tok,"@");codigo();
    if (pila<0) return(error(7));
    mem=cuenta;
    return(ntok);
}

/*=====
   En este procedimiento se coloca el codigo intermedio que
   se genera al compilar.
   ===== */

int parser::codigo(void)
{
    strcpy(probar1->p,tok);
    mem++;probar1++;
    return(0);
}

/*+++++
   AQUI TERMINA LA IMPLEMENTACION DEL OBJETO PARSER
   +++++ */

```

Objetivo : Realizar un editor con características similares a los editores comerciales.

```
/*+++++
AQUI INICIA LA IMPLEMENTACION DEL OBJETO EDITOR
+++++*/
```

```
void editor::pre(void)
{
    register int s;
    char buf[4096];
    cc(0,3); clrscr(); gotoxy(1,1); cc(0,7);
    cputs(" Editor      Evaluar      Graficar      Tutor      \n");
    gotoxy(16,25); cputs(" Ayuda F1  Menu F10  Salir ALT-X  ");
    gotoxy(56,25); cputs("R:      C:      "); cc(0,3); border(1,2,80,24);
    gettext(1,1,80,25,buf); border(18,7,62,17);
    gotoxy(21,9); cputs("Universidad Nacional Autonoma de México");
    gotoxy(29,11); cputs("Facultad de Ingeniería");
    gotoxy(33,13); cputs(" SISTEMA-TUTOR ");
    gotoxy(33,15); cc(1,6); cputs(" M A T U N A M \n");
    s=toascii(getch()); if(s==0) getch();
    puttext(1,1,80,25,buf); gotoxy(1,25); cc(1,6); cputs(" SISTEMA-TUTOR ");
    gotoxy(71,25); cputs(" R A D C "); cc(0,3); getcwd(tit,MAXPATH);
    if (strlen(tit)>3) strcat(tit,"\\");
    strcat(tit,"NINGSUNO.MAT");
    gotoxy((80-strlen(tit))/2,2); printf(" %s ",tit);
}

int editor::leenom(int x1, int y1)
{
    int x2=x1;
    int a=1,i=0;
    gotoxy(x2,y1-1); cputs("Nombre del archivo:");
    gotoxy(x2,y1);
    while (a) {
        int s=toascii(toupper(getch()));
        switch(s) {
            case 27: a=0; i=1; break;
            case 13: nomf[i]='\0'; a=0; i=0; break;
            case 8: i--; if(i<0) i++;
                    x2--; if(x2<x1) x2++;
                    gotoxy(x2,y1); cprintf(" "); break;
            case 0: getch(); break;
            default: gotoxy(x2,y1); cprintf("%c",s); x2++;
                    nomf[i++]= (char)s;
        }
        gotoxy(x2,y1);
    }
    return(i);
}
```

```

int editor::cargar(void)
{
    int w=0, v=3, v1,w1;
    char buf[2*41*6];
    FILE *in;
    v1=(80-strlen(tit))/2; w1=strlen(tit); gettext(20,10,60,15,buf);
    cc(0,7); lvp(20,10,60,15); border(20,10,60,15); cc(1/BLINK,2);
    gotoxy(35,10);cputs(" Cargando ");cc(0,7);strcpy(s1,tit);
    if (strlen(tit)!=0) {strcpy(tit,s1);puttext(20,10,60,15,buf);cc(0,3);
    return(0);}
    flags=fnsplit(nom,drive,dir,file,ext);
    if (!(flags & EXTENSION))
        strcat(nom,".MAT");
    if (flags & DRIVE) strcpy(tit,nom);
    else { getcwd(tit,MAXPATH);
        if (strlen(tit)>3) strcat(tit,"\\");
        strcat(tit,nom); }
    if((in=fopen(tit,"rt"))==NULL) {
        fclose(in);strcpy(tit,s1);tipos(1);puttext(20,10,60,15,buf); return(1);
    }
    else {
        cc(0,3); puttext(20,10,60,15,buf); lvp(1,2,80,24); k=fin=inicio;lim():
        while (!feof(in)){
            fgets(linea,80,in);
            if (ferror(in)){strcpy(tit,s1);fclose(in); tipos(1); return(1); }
            linea[strlen(linea)-1]='\0'; fin=k; strcpy(k->1,linea); k++;
            if (v<=23) { gotoxy(2,v++);cputs(linea); } lim();
        }
        fclose(in);
        for (w=0; w<w1+2; w++) { gotoxy(v1++,2);printf("%c",205); }
        for (w=0; w<strlen(tit); w++) tit[w]=toupper(tit[w]);
        gotoxy((80-strlen(tit))/2,2);printf(" %s ",tit);
        k=inicio;strcpy(linea,k->1);x=2;y=3;gotoxy(x,y);
    }
    return(0);
}

```

```

int editor::salvar(void)
{
    char buf[2*41*6];
    int w=1,s,v1,w1;
    FILE *in;
    v1=(80-(w1=strlen(tit)))/2;
    gettext(20,10,60,15,buf); cc(0,7); lvp(20,10,60,15);
    border(20,10,60,15);cc(1/BLINK,2);
    flags=fnsplit(tit,drive,dir,file,ext);
    if (strcmp(file,"NINGUNO")==0) {
        chequeo();cc(0,7);gotoxy(25,13);cputs(" o Cambiar nombre ");
        gotoxy(45,14);cputs("< S / C >");
        while (w) {
            s=toupper(getch());
            switch (s) {

```

```

case 8:w=0;break;
case 67:lvp(20,10,60,15);
    if (leenom(21,13)!=0)
        {puttext(20,10,60,15,buf);return(1);}
    flags=fnsplit(nom,drive,dir,file,ext);
    if (!(flags & EXTENSION))
        strcat(nom, ".MAT");
    if (flags & DRIVE) strcpy(tit,nom); else
    {getcwd(tit,MAXPATH);
    if (strlen(tit)>3) strcat(tit, "\\");
    strcat(tit,nom);}
    w=0;break;
case 0 :getch();break;
case 27:puttext(20,10,60,15,buf);return(1);
}
}
gotoxy(35,10);cputs(" Salvando ");cc(0,7);lvp(20,10,60,15);
gotoxy(25,13);cputs(tit);cc(0,3);
if((in=fopen(tit,"rt"))==NULL) {
    fclose(in);
    if((in=fopen(tit,"wt"))==NULL) {
        fclose(in);tipos(2);puttext(20,10,60,15,buf);return(2);
    }
    for (k=inicio; k<=fin; k++)
    {
        strcpy(linea,k->1); strcat(linea, "\n\r"); fputs(linea,in);
        if
(ferror(in)){fclose(in);tipos(2);puttext(20,10,60,15,buf);return(2);}
    }
    fclose(in);
}
else {
    fclose(in);
    flags=fnsplit(tit,drive,dir,file,ext); strcpy(ext, ".BAK");
    if (flags & DIRECTORY && (strlen(dir)!=1)) strcpy(dir, "");
    fnmerge(s1,drive,dir,file,ext);
    w=back(tit,s1);
    if (w==0) {
        if((in=fopen(tit,"wt"))==NULL) {
            fclose(in);tipos(2);puttext(20,10,60,15,buf);return(2);
        }
        for (k=inicio; k<=fin; k++)
        {
            strcpy(linea,k->1); strcat(linea, "\n\r"); fputs(linea,in);
            if (ferror(in))
                {fclose(in);tipos(2);puttext(20,10,60,15,buf);return(2);}
        }
        fclose(in);
    }
    else
        tipos(w);
} puttext(20,10,60,15,buf);

```

```

for (w=0; w<w1+2; w++) { gotoxy(v1++,2);printf("%c",205); }
for (w=0; w<strlen(tit); w++) tit[w]=toupper(tit[w]);
gotoxy((80-strlen(tit))/2,2);printf(" %s ",tit); return(0);
}

```

```

void editor::coor(int r)
{
cc(0,7); gotoxy(58,25); cputs(" ");
gotoxy(58,25); cprintf("%d",r); cc(0,3);
}

```

```

void editor::cooc(int c)
{
cc(0,7); gotoxy(66,25); cputs(" ");
gotoxy(66,25); cprintf("%d",i--c); cc(0,3);
}

```

```

void editor::ruido(void)
{
sound(733); delay(10); nosound();
};

```

```

void editor::lim(void)
{
register int i;
for (i=0; i<80; i++) linea[i]='\0';
}

```

```

void editor::mueve(void)
{
for (mover=inicio; mover<=fin; mover++) {
if (e>=yf) break;
gotoxy(2,e);cputs(bla);
gotoxy(2,e++);cputs(mover->1);
}
}

```

```

void editor::muevek(void)
{
for (mover=k; mover<=fin; mover++) {
if (e>=yf) break;
gotoxy(2,e);cputs(bla);
gotoxy(2,e++);cputs(mover->1);
}
}

```

```

void editor::mueva20(void)
{
for (mover=fin-20; mover <=fin; mover++) {
if (e==yf) break;
gotoxy(2,e);cputs(bla);
gotoxy(2,e++);cputs(mover->l);
}
}

```

```

void editor::alicar(void)
{
k++;
for (mover=fin; mover>=k; mover--) {
strcpy((mover+1)->l,mover->l);
} e=2; fin++;
for (mover=k+1; mover<=fin; mover++) {
if (y+e==yf) break;
gotoxy(2,y+e);cputs(bla);
gotoxy(2,y+e++);cputs(mover->l);
}
y++;cl++;
if (y==yf) { y--;vid(1); }
gotoxy(2,y);cputs(bla);lim();strcpy(k->l, linea);
}

```

```

void editor::siglinea(char *plinea, int w)
{
if(k!=fin) {
k++;
if((strlen(k->l)+w+1) < 76) {
strcat(plinea," ");strcat(plinea,k->l);strcpy(k->l,plinea);
e++;y;muevek();
if(y==yf) {y--;vid(1);gotoxy(2,y);cputs(bla);
gotoxy(2,y);cputs(k->l); }
lim(); strcpy(plinea,k->l);
}
else {
k--;
if(insert) {
alicar();strcpy(k->l,plinea);e=y;muevek();
gotoxy(2,y);cputs(plinea);lim();strcpy(plinea,plinea);
}
else
plinea[0]='\0';
}
}
else {
alicar();strcpy(k->l,plinea);
gotoxy(2,y);cputs(plinea);lim();strcpy(plinea,plinea);
}
}
}

```

```

int editor::finlinea(int h, int v, int vl)
{
int yo=h;char parte[80];
if(isalnum(linea[h])!=0) {
while(isalnum(linea[--h])!=0 && h>0);
if(h==0) {
tipos(3); linea[76]='\0'; strcpy(parte, linea); strcpy(k->l, linea);
gotoxy(2, y); cputs(bla); gotoxy(2, y); cputs(k->l);
}
else {
int nada=h, avanza=0;
for(h=nada+1; h<=yo; h++)
parte[avanza++]=linea[h];
parte[avanza]='\0'; linea[nada+1]='\0';
gotoxy(2, y); cputs(bla); gotoxy(2, y); cputs(linea);
strcpy(k->l, linea);
h=strlen(parte);
if(v1==79) { parte[h++]= (char)v; parte[h]='\0'; }
siglinea(parte, h);
if(insert) return(h);
}
}
else {
if(linea[h]!=' ') {
parte[0]=linea[h];
parte[1]=(char)v;
parte[2]='\0';
linea[h]='\0';
gotoxy(2, y); cputs(bla); gotoxy(2, y); cputs(linea);
strcpy(k->l, linea);
h=strlen(parte);
siglinea(parte, h);
}
else {
if(insert && v1==79) {
parte[0]=(char)v;
parte[1]='\0';
linea[h]='\0';
gotoxy(2, y); cputs(bla); gotoxy(2, y); cputs(linea);
strcpy(k->l, linea);
h=strlen(parte);
siglinea(parte, h);
}
else {
linea[h]='\0';
gotoxy(2, y); cputs(bla); gotoxy(2, y); cputs(linea);
strcpy(k->l, linea); parte[0]='\0';
if(insert && k!=inicio) k++;
}
}
}
return(strlen(parte));
}
}

```



```

int editor::selmenu(void)
{
    Mod menu(principal, 2. 1);
    return(menu.selecciona());
}

void editor::vid(int t)
{
    char npan[4096];
    if (t==1) { gettext(2,4,79,23,npan); puttext(2,3,79,22,npan); }
    else { gettext(2,3,79,22,npan); puttext(2,4,79,23,npan); }
}

void editor::desplega(void)
{
    int xi=1,yi=2,xf=80,tecla=1,yf=24;
    int j=1,c;cl=1;insert=0;y=3;x=2;
    inicio=(largo *)malloc(100*sizeof(largo));
    fin=k=inicio;
    getcwd(rmu,MAXPATH); if(strlen(rmu)>3) strcat(rmu,"\\");
    strcpy(s1,rmu); strcat(s1,"entrada.exe");
    spawnl(P_WAIT,s1,"entrada","MATUNAM",NULL);_setcursortype(0);
    pre(); coor(cl); cooc(x); gotoxy(x,y);_setcursortype(1);
    for (i=0; i<78; i++) bla[i]=' '; bla[i]='\0'; lim();
    while (tecla) {
        j=toascii(getch());
        switch (j) {
            case CONTROL:
                j=toascii(getch());
                switch(j) {
                    case Flecha_Arriba:
                        strcpy(k->1, linea);lim();
                        if(y>yi)
                            if (y==yi+1) {
                                if (k>=inicio) {
                                    k--;vid(2);strcpy(linea,k->1);
                                    gotoxy(2,3);cputs(bla);
                                    gotoxy(2,3);cputs(k->1);cl--;
                                }
                                else {
                                    if (k<=inicio) k=inicio;ruido();
                                }
                            }
                        else {
                            if (k<=inicio) ruido();
                            else {
                                --k;strcpy(linea,k->1);y--;cl--;
                            }
                        }
                        strcpy(linea,k->1);
                        coor(cl);break;

```

```

case Flecha_Abajo:
    strcpy(k->l, linea); lim();
    if (y<yf)
        if (y==yf-1) {
            if (k<fin) {
                k++; vid(1); strcpy(linea, k->l);
                gotoxy(2,23); cputs(bla);
                gotoxy(2,23); cputs(k->l); cl++;
            }
            else {
                if (k>=fin) k=fin; ruido();
            }
        }
        else {
            if (k==fin) ruido();
            else {
                ++k; strcpy(linea, k->l); y++; cl++;
            }
        }
        strcpy(linea, k->l);
        cor(cl); break;
case Flecha_Izquierda:
    if (x>xi)
        if (x==xi+1) ruido(); else x--; cooc(x); break;
case Flecha_Derecha:
    if (x<xf)
        if (x==xf-1) ruido(); else x++; cooc(x); break;
case Home:
    x=xi+1; cooc(x); break;
case End:
    x=strlen(linea)+2; cooc(x); break;
case Ins:
    if (insert) { _setcursortype(1);
        insert=0; cc(0,7); gotoxy(65,1); cputs("  "); cc(0,3);}
    else {
        setcursortype(2); insert=1; cc(0,7); gotoxy(65,1);
        cputs("INS"); cc(0,3);
    } break;
case Del:
    if (strlen(linea)!=0) {
        for (c=x-2; c<=strlen(linea); c++) {
            linea[c]=linea[c+1];
            gotoxy(c+2,y); cprintf("%c", linea[c]);
        }
        gotoxy(c+1,y); cputs(" ");
    } break;
case Pagina_Arriba:
    strcpy(k->l, linea); lim();
    if (k==inicio) {
        k=inicio; cl=1; y=3;
    }
    else {

```

```

e=(k-inicio)+1;
if (e<=21 && e>=1) {
    cl=1; y=e=3;
    mueve();
    k=inicio;
}
else {
    cl=21;
    e=(k-inicio-21)+1;
    if (e<=21 && e>=1) {
        e=y=3;k=21;
        muevek();
    }
    else {
        e=y-2; c=23-(21-e);
        k=e;e=23;
        for (mover=k; mover>=inicio; mover--){
            if (e==c) k=mover;
            if (e<=yi) break;
            gotoxy(2,e);cputs(bla);
            gotoxy(2,e--);cputs(mover->1);
        }
        if (e>yi)
            while(e>yi) {
                gotoxy(2,e--);cputs(bla);
            }
    }
}
}
}
strcpy(linea,k->1);
coord(cl);break;
case Pagina_Abajo:
strcpy(k->1,linea);lim();
if (k==fin) {
    k=fin; cl=abs(fin-inicio)+1;
}
else {
    if (inicio>=fin-20) {
        for (cl=1,e=3,mover=inicio; mover <fin; mover++){
            cl++;e++;
        }
        k=fin;y=e;
    }
    else {
        e=3;
        if (k>=fin-20 && k<=fin) {
            mueve20();
            k=fin;cl=abs(fin-inicio)+1; y=23;
        }
        else {
            e=(23-y)+1;c=y-2;
            cl+=21;k+=e;e=3;
        }
    }
}

```

```

        for (mover=k; mover<=fin; mover++) {
            if (e==c) k=mover;
            if (e>yf) break;
            gotoxy(2,e);cputs(bla);
            gotoxy(2,e++);cputs(mover->l);
        }
        if (e<yf && e!=3)
            while(e<yf) {
                gotoxy(2,e++);cputs(bla);
            }
    }
}
strcpy(linea,k->l);
coor(c1);break;
case Control_Home:
strcpy(k->l,linea);
e=3; mueve();
k=inicio;lim();strcpy(linea,k->l);
x=x+1;y=y+1;coor(c1=1);cooc(x);break;
case Control_End:
strcpy(k->l,linea);e=(fin-inicio)+1;
if (e<=21 && e>=1) {
    e=3; mueve();
    y=-e;
}
else {
    e=3;mueve20();
    y=-e;
}
k=fin;lim();strcpy(linea,k->l);
x=x+1;c1=abs(fin-inicio);coor(++c1);cooc(x);break;
case F1:
strcpy(k->l,linea);ayuda ayu;_setcursortype(0);
ayu.ini_ayuda();lim();strcpy(linea,k->l);
if(insert) _setcursortype(2); else _setcursortype(1);
break;
case F2:
strcpy(k->l,linea);salvar();lim();strcpy(linea,k->l);
break;
case F3:
strcpy(k->l,linea);cargar();lim();strcpy(linea,k->l);
break;
case F10:
strcpy(k->l,linea); int nuevo=0;nuevo=menu();
if (k==inicio) { c1=1;x=2;y=3;coor(c1);cooc(x); }
if (nuevo==4) { lim();strcpy(k->l,linea); }
lim();strcpy(linea,k->l);
break;
case Alt_X:
strcpy(k->l,linea); int a=1,s; char bb[2*41*6];
gettext(20,10,60,15,bb);

```

```

chequeo();while(a) { s=toascii(toupper(getch()));
switch(s) {
case 83:salvar();
case 78:cc(7,0);clrscr();free(inicio);_setcursortype(2);
exit(0);
case 0 :getch();
case 27:puttext(20,10,60,15,bb);a=0; break;
});
};
break;
case Return:
strcpy(k->l, linea);
if(insert) {
alicar();strcpy(linea, (k-1)->l);
if(strlen(linea)>x-2) {
i=0;
for (c=x-2;c<=strlen(linea);c++)
k->l[i+1]=linea[c];k->l[i]='\0';
linea[x-2]='\0';strcpy((k-1)->l, linea);
gotoxy(2,y-1);cputs(bla);gotoxy(2,y-1);cputs(linea);lim();
gotoxy(2,y);cputs(bla);gotoxy(2,y);cputs(k->l);
}
strcpy(linea,k->l);
}
else {
if (k==fin) { k++;fin=k;y++;cl++;
if (y==yf) { y--;vid(1); }
gotoxy(2,y);cputs(bla);lim();strcpy(k->l, linea); }
}
x=xi+1;coor(cl);cooc(x);break;
case Backspace:
for (c=x-3; c<=strlen(linea); c++) {
linea[c]=linea[c+1];
gotoxy(c+2,y);printf("%c",linea[c]);
}
if (c)=0 && x!=xi+1) {gotoxy(c+1,y);cputs(" ");}
x--;strcpy(k->l, linea);
if (x==xi) { y--;
if (y<=yi) {
ruido();y++;x++; }
else {
if(strlen(linea)==0) {
if(k==fin) fin--;
else {
for (mover=k; mover<=fin; mover++)
strcpy(mover->l, (mover+1)->l);e=y+1;fin--;
muevek();
if (mover>fin && e<yf) { gotoxy(2,e);cputs(bla); }
}
}
}
else {
if((strlen((k-1)->l)+strlen(linea))<76) {

```

```

        strcat((k-1)->l, linea);
        for (mover=k; mover<=fin; mover++)
            strcpy(mover->l, (mover+1)->l); e=y+1; fin--;
        muevek();
        if (mover>fin && e<yf) { gotoxy(2,e); cputs(bla); }
        gotoxy(2,y); cputs(bla); gotoxy(2,y); cputs((k-1)->l);
    }
    cl--; k--; lim(); strcpy(linea, k->l); x=strlen(linea)+2; coor(cl); }
} cooc(x); break;
case Tab:
for (c=strlen(linea)+1; c>=0; c--) linea[c+4]=linea[c];
for (c=1; c<=4; c++){ linea[x-2]= ' '+c; }
cputs(linea); cooc(x); break;
case Control_Y:
if(fin<=inicio) {
    gotoxy(2,y); cputs(bla); fin=inicio; lim(); strcpy(k->l, linea);
}
else {
    if (k==fin) {
        gotoxy(2,y); cputs(bla); fin--; k=fin; cl--; coor(cl); y--;
        if (y==yi) { y++; gotoxy(2,y); cputs(k->l); }
    }
    else {
        for (mover=k; mover<=fin; mover++)
            strcpy(mover->l, (mover+1)->l); e=y; fin--;
        muevek();
        if (mover>fin && e<yf) { gotoxy(2,e); cputs(bla); }
    }
}
strcpy(linea, k->l); break;
case Esc:
break;
default:
if(x<79) {
    if (insert) {
        if ((c=strlen(linea)-1)<76) {
            for (c=strlen(linea)+1; c>=x-2; c--) linea[c+1]=linea[c];
            linea[x-2]= (char) j;
            if(linea[x-3]=='\0') { c=x-3;
                while(linea[c]!='\0' && c>=0) linea[c--]=' ';
            }
            gotoxy(2,y); cputs(linea);
        }
    }
    else {
        finlinea(C, j, x);
        if(k!=inicio) {
            k--; y--; strcpy(linea, k->l);
        }
    }
}

```

```

for (c=strlen(linea)+1; c>=x-2; c--) linea[c+1]=linea[c];
linea[x-2]=(char)j;
if (linea[x-3]!='\0') { c=x-3;
while (linea[c]!='\0' && c>=0) linea[c--]=' ';
}
gotoxy(2,y);cputs(linea);
}
}
if (!insert) {
linea[x-2]=(char)j;
gotoxy(x,y);cprintf("%c",j);
}
x++;
if (x==xf){ y++;cl++;x=xi+1; } coor(cl);cooc(x);
if (linea[i-3]!='\0') { c=i-3;
while (linea[c]!='\0' && c>=0) linea[c--]=' ';
}
}
else {
x=xi+1+finlinea(x-3,j,x);coor(cl);cooc(x);
}
}
gotoxy(x,y);
}
}
}
/*+++++
AQUI TERMINA LA IMPLEMENTACION DEL OBJETO EDITOR
+++++*/

```

Pruebas del sistema MAT-UNAM.

Durante las etapas anteriores, se ha tratado de construir una aplicación de software, partiendo de un control abstracto hasta llegar a una implementación tangible. Es en este punto donde entran las pruebas, para tratar de encontrar fallas en el sistema.

Cuando se aplica una prueba al software implementado, se busca cubrir los siguientes objetivos:

- 1.- La prueba es un proceso de ejecución de un programa con la intención de descubrir un error.
- 2.- Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces.
- 3.- Una prueba tiene éxito si descubre un error no detectado hasta entonces.

Si la prueba se lleva a cabo con éxito, descubrirá errores en el software. Además, la prueba demuestra hasta que punto las funciones del software parecen funcionar de acuerdo con las especificaciones.

Además, los datos que se van recogiendo a medida que se llevan a cabo las pruebas, proporcionan un indicador de la confiabilidad del software. La prueba no puede asegurar la ausencia de defectos, sólo puede demostrar que existen defectos en el software.

En el sistema MAT-UNAM, se aplicaron las pruebas con casos típicos, que validaran las entradas, los ciclos y comparaciones. Dichas pruebas se aplicaron por unidad, integración y sistema total.

Esto es, al referirnos por unidad, la prueba se aplica a cada uno de los procedimientos de cada módulo. Por integración, cuando se interrelaciona un módulo con otro. Finalmente, el sistema total como un todo.

CAPITULO V
CONCLUSIONES

En este capítulo se describirán las conclusiones obtenidas durante el desarrollo del sistema Mat-UNAM. Así como, algunas alternativas para un apoyo adecuado a los sistemas tutores.

Primeramente, podemos afirmar que el objetivo inicial, se cumplió de acuerdo a lo esperado. Pero debemos decir de igual manera que surgieron una infinidad de problemas a través del desarrollo.

Dichos problemas se debieron principalmente por las características del software seleccionado, para el desarrollo del sistema. Dentro de las fallas que presentó dicho software se tienen las siguientes:

- Restricción en el manejo de memoria (máximo 64 Kb). Esto es, no se puede acceder memoria con apuntadores de tipo far, con lo cual se limitan mucho algunos procesos que requieren el uso de la memoria.
- Manejo indecuado de archivos. En este sentido el software utiliza un buffer para el manejo de archivos, pero al terminar de usar algún archivo el software no libera completamente la memoria que utiliza dicho buffer.

Por lo anterior, se tuvo que hacer un manejo muy delicado con la memoria para poder optimizarla. Pero también, se ha conseguido

obtener un producto final con tutoría, sobre los aspectos que se pretendían. Dichas tutorías se avocaron sobre raíz cuadrada, factor común y despeje, derivada parcial de un polinomio, y Álgebra básica.

En todas las mencionadas anteriormente, se indica paso a paso la solución del problema propuesto por el usuario final (alumno y/o maestro). La tutoría es uno de los aspectos que se tenían contemplados en el desarrollo, y que otros sistemas de Álgebra computacional no lo contemplan.

Además, se ha tomado en cuenta una parte adicional en la zona de tutoría. Esta consiste en mencionar los errores más comunes, que el estudiante de Álgebra tiene. En dicha sección, se indica el error que se comete y la forma en que debe corregirse.

Por otra parte, una de las mejoras que pudieran implementarse al sistema actual es la adición del modo gráfico en todas y cada una de sus partes, para resaltar la presentación y estimulación hacia el usuario final de dicho paquete.

La orientación de los paquetes educativos como éste, deben ser dirigidos hacia una educación de mejor calidad, pero sin descuidar la interrelación de alumnos y profesores. Podemos concluir entonces, que el presente trabajo es un nexo entre ambos para la obtención de tal fin.

Finalmente, esperamos que éste tipo de desarrollos se sigan dando, y sobre todo que exista la continuidad de ellos, ya que, al término de un desarrollo de tesis, éste por lo general es olvidado.

BIBLIOGRAFIA

- REVISTA BYTE JULIO DE 1980
VOL 5 No. 7
EDITORIAL Mc. GRAW-HILL.

- REVISTA BYTE JUNIO DE 1984
VOL 9 No. 6
EDITORIAL Mc GRAW-HILL.

- OBJECT ORIENTED PROGRAMMING WITH QUICK PASCAL
NAMIR SHAMMAS.
ED. JOHN WILEY & SONS, INC. 1990.

- OBJECT ORIENTED PROGRAMMING GUIDE REFERENCE
TURBO PASCAL 5.5
BORLAND INTERNATIONAL INC.
1989.

- THE ART OF COMPUTER PROGRAMMING
DONALD D. KNUTH
ED. PRENTICE HALL

- EUREKA: THE SOLVER OWNER'S HANDBOOK.
BORLAND INTERNATIONAL INC.
1987.

- MICROSOFT MUMATH SYMBOLIC MATHEMATICS PACKAGE
MICROSOFT CORPORATION AND SOFT WAREHOUSE.
1983.

- REVISTA MAGISTERIO No. 217
SINDICATO NACIONAL DE TRABAJADORES DE LA EDUCACION.
MEXICO 1983. P.P. 19-21.

- PERIODICO EL MAESTRO. JULIO DE 1986
AÑO III. SEPTIMA EPOCA. No. 39
P.P. 5.

- AN INTRODUCTION TO ENGINEERED SOFTWARE.
GILLETT WILL D.
POLLACK SEYMOUR V.
ED. HOLT, RINEHART AND WINSTON.
NEW YORK, 1982.

- SCIENTIFIC PASCAL.
FLANDERS HARLEY.
ED. PRENTICE HALL
1984.

- SMALLTALK TUTORIAL AND PROGRAMMING HANDBOOK.

DIGITALK INC.

1988.

- SYSTEMS DEVELOPMENT A PRACTICAL APPROACH.

WILLIAM AMADIO.

MITCHELL PUBLISHING INC.

1989.

- INGENIERIA DE SOFTWARE.

ROGER S. PRESSMAN.

ED. MCGRAW-HILL

1989.

- COMPUTER ALGEBRA. SYSTEMS AND ALGORITHMS FOR

ALGEBRAIC COMPUTATION.

J. H. DAVENPORT

ED. ACADEMIC PRESS

1988.

- COMPUTER GRAPHICS AND CAD FUNDAMENTALS

NOEL M. MORRIS

ED. PITMAN

1986.

- ALGEBRA AND TRIGONOMETRY

MAX A. SOBEL

NORBERT LERNER

ED. PRETINCE HALL

1991.

APENDICE A

MANUAL DE USUARIO

El presente manual tiene la finalidad de proporcionar una guía y ayuda al usuario, sobre el uso y manejo del sistema MAT-UNAM.

Este manual está dividido en tres partes para dar un panorama más claro sobre cada aplicación del sistema MAT-UNAM. La primera de ellas consiste en indicar los requerimientos de hardware, así como el manejo del teclado e inicio de MAT-UNAM.

La segunda de ellas consiste en el conocimiento general de las partes en que consiste el sistema MAT-UNAM, así como también, las aplicaciones.

La tercera sección consiste en indicar la sintaxis de cada una de las funciones, y errores de compilación de MAT-UNAM.

¿Que es MAT-UNAM?

MAT-UNAM es un paquete de algebra computacional que surge como apoyo para el autoaprendizaje del estudiante de matemáticas al crearse una area denominada de tutoría. Pero también, la facilidad de obtener gráficas de las funciones definidas por el usuario.

MAT-UNAM contiene las funciones básicas de trigonometría, permite realizar conversiones de tipo de coordenadas, evaluar integrales definidas, derivación y factorización.

PARTE I

Requerimientos de hardware.

MAT-UNAM puede ejecutarse en cualquier equipo que cumpla con las siguientes características:

- + Computadora PC con dos unidades de disco flexible, o una con disco duro y una unidad de disco flexible.

- + 640 Kb de memoria.

- + Tarjeta de video gráfica de alguno de los tipos:
 - CGA
 - HERCULES
 - EGA
 - VGA

MAT-UNAM contiene los siguientes archivos:

- + MATUNAM.EXE
- + MATUNAM.AYU
- + ENTRADA.EXE
- + RACC.EXE
- + GRAFICAR.EXE
- + ABCMAT.EXE

Se recomienda hacer una copia de respaldo de los archivos en otro disco antes de empezar a usar MAT-UNAM.

Como ya se menciono MAT-UNAM puede ejecutarse en disco duro o en disco flexible. Así pues, si desea trabajar con MAT-UNAM en disco duro se recomienan los siguientes pasos:

1) Crear un subdirectorío en la unidad C con el nombre MATUNAM, como sigue:

```
C:\> MD MATUNAM <ENTER>
```

2) Copiar los archivos de MAT-UNAM en el subdirectorío MATUNAM que se acabo de crear en el paso anterior como sigue:

```
C:\> COPY A:*. * C:\MATUNAM <ENTER>
```

A partir de este momento ya podemos iniciar a trabajar con MAT-UNAM. Si deseamos trabajar en disco duro, primero debemos cambiarnos al subdirectorío MATUNAM.

```
C:\> CD MATUNAM <ENTER>
```

Teclear GRAPHICS y presionar enter (este comando es del sistema operativo)

```
C:\> GRAPHICS <ENTER>
```

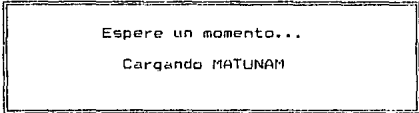
Ahora tecleamos MATUNAM y presionamos la tecla de enter.

```
C:\> MATUNAM <ENTER>
```

Para trabajar con unidad de disco flexible, también presionar GRAPHICS del sistema operativo, cambiamos a la unidad en cuestión, tecleamos MATUNAM y presionamos la tecla de enter.

```
A:\> MATUNAM <ENTER>
```

MAT-UNAM empezará a ejecutarse y aparecerá en la pantalla un mensaje que le indicará que MAT-UNAM está preparándose para iniciar como se ve en la siguiente figura:



Espere un momento...
Cargando MATUNAM

Inmediatamente aparecerá en la pantalla una gran cantidad de números de diferente tamaño y color. Esta permanecera cambiando de color el fondo del video y los números hasta que sea presionada alguna tecla. La figura 2 muestra dicha pantalla. Es importante hacer notar que si no se cuenta con tarjeta de video gráfica, no se activará la pantalla mencionada.

Posteriormente, aparecerá una presentación como se ve en la figura 3 y al igual que con la pantalla anterior, si no se cuenta con tarjeta de video gráfica, tampoco se verá. Para continuar deberá presionar alguna tecla.

Entonces ya estaremos en el editor de MAT-UNAM, el cual mostrará la identificación del paquete, esperando también que sea presionada una tecla. La forma de entrada del editor se muestra a continuación:

Editor Evaluar Graficar Tutor

Universidad Nacional Autónoma de México
Facultad de Ingeniería
SISTEMA-TUTOR
M A T U N A M

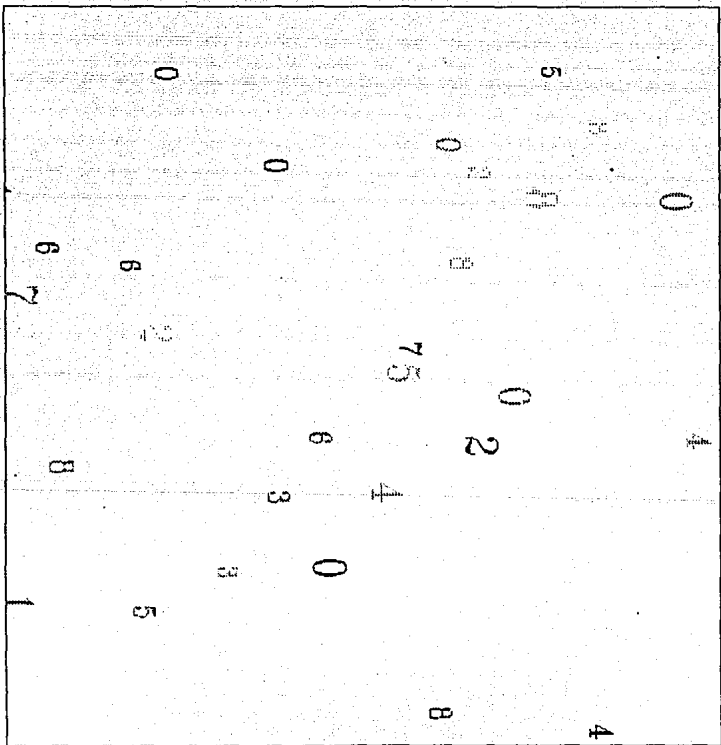


Figura 2.

Universidad Nacional Autónoma de México

Facultad de Ingeniería



Sistema



Mat-UNAM

Figura 3.

En este momento ya podemos iniciar a trabajar con MAT-UNAM. El editor se encuentra en modo sobreescritura por default, si se desea cambiar a modo inserción sólo se debe presionar INS (INSERT en otro tipo de computadora). Para regresar al modo anterior presionamos nuevamente la tecla INS.

La pantalla de edición está constituida por tres zonas :

- + LINEA DE ESTADO
- + EDICION
- + MENU

Zona de línea de estado.

La línea de estado esta colocada en la parte inferior de la pantalla, indica las acciones que pueden ser tomadas en la edición, como son la ayuda, ver en que renglón y columna se encuentra el cursor, acceso al menu principal, o salir del sistema MAT-UNAM.

Zona de edición.

La zona de edición puede considerarse como una de las más importantes, porque aquí es donde se introducen los datos necesarios para que MAT-UNAM realice las operaciones que se le indiquen.

Esta zona se divide en dos zonas más, la primera denominada zona de declaración de funciones de usuario, y la segunda de ejecución de funciones.

La zona de declaración de funciones es donde el usuario define las funciones que el desea sean evaluadas o gráficas, la sintaxis de esta zona es:

```
; NOMBRE DE FUNCION ( VARIABLES ) = FUNCION ;
```

donde:

NOMBRE DE FUNCION : es el nombre con el cual MAT-UNAM reconocerá a la función (el nombre tiene un máximo de 10 caracteres).

VARIABLES : es la lista de variables separadas por una coma (el número máximo de variables es de 3).

FUNCION : es la definición formal de la función.

El usuario puede definir 10 funciones como máximo. Un ejemplo de como se definiría una función es la siguiente:

```
; f ( x, y ) = x^2 + 5*x*y^3 ;
```

La zona de ejecución es donde se indicará a MAT-UNAM, los tipos de operaciones que han de aplicarse a las funciones definidas por el usuario. Por ejemplo:

```
sen(f)    <== se obtendrá el seno de la función f.  
abs(x)    <== se obtendrá el valor absoluto de x.  
raiz2(4)  <== se obtendrá la raíz cuadrada de 4.
```

Movimientos en el editor.

Un aspecto importante es saber como podemos movernos a través del editor, así como también que teclas de función nos permite el paquete en que trabajaremos.

MAT-UNAM tiene un editor con las características similares a la de los editores comerciales, pero no en su totalidad. MAT-UNAM requiere un editor de manejo sencillo para el usuario. Así pues, el editor se limitó de acuerdo a la aplicación a la que esta orientada MAT-UNAM.

Una línea en MAT-UNAM tiene un tamaño máximo de 80 caracteres. El programa puede tener un máximo de 100 líneas. El movimiento a través del editor de MAT-UNAM se presenta en la siguiente tabla:

Movimiento del cursor	Tecla
Cáriter izquierdo	Flecha izquierda
Caracter derecho	Flecha derecha
Línea arriba	Flecha arriba
Línea abajo	Flecha abajo
Página arriba	PgUp
Página abajo	PgDn
Inserción	Ins
Inicio de línea	Home
Fin de línea	End
Borrar línea	Ctrl-Y
Borrar caracter izquierdo	Backspace
Borrar caracter	Del
Inicio de archivo	Ctrl-Home
Fin de archivo	Ctrl-End

Tabla 1 Movimiento del cursor.

Además, existen algunos comandos adicionales como son:

F1 Ayuda
F2 Salvar archivo
F3 Cargar archivo
F10 Menú principal
Alt-X Salida del paquete
ESC Abortar la última operación.

Zona de menú.

La zona de menú es la tercera y última zona de la pantalla de edición. Ésta se ubica en la parte superior de la pantalla. La zona de menú es la que nos indica las opciones que podemos seleccionar para realizar alguna operación en especial.

Esta zona esta constituida por las siguientes opciones:

+ EDITOR
+ EVALUAR
+ GRAFICAR
+ TUTOR

Las opciones mencionadas se describen con detalle en la siguiente sección.

PARTE II

Conocimiento general de MAT-UNAM.

El conocimiento general del sistema MAT-UNAM esta descrita en esta sección. Se tratará con detenimiento cada una de las opciones de la zona de menú mencionada en la primera parte.

La zona de menú se encuentra constituida por cuatro opciones, la primera de ellas es la de EDITOR. Esta a su vez se constituye o subdivide en más opciones, que a continuación se mencionan y describen.

- + CARGAR
- + SALVAR
- + SALVAR COMO
- + NUEVO
- + CAMBIAR DIR
- + DIRECTORIO
- + SALIR

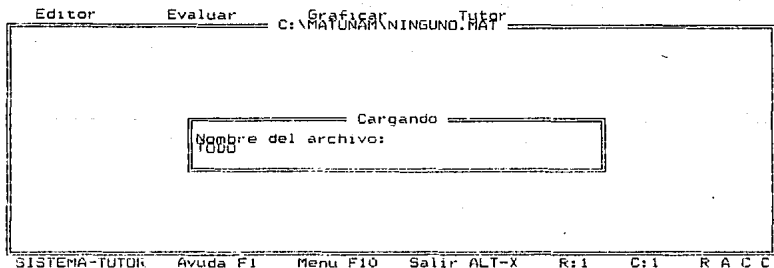
Cargar.

Esta opción sirve para cargar en el editor algún archivo creado por MAT-UNAM con anterioridad. La opción solicitará el nombre del archivo que se desea cargar en el editor. Si no se le proporciona la extensión del archivo MAT-UNAM asumirá la extensión .MAT.

Además, MAT-UNAM asume que el archivo que se desea cargar se encuentra en la unidad y ruta actual. Así pues, si se desea cargar el archivo TODO.MAT, en el editor, se debe seleccionar primero el menú principal presionando F10.

Ahora seleccionamos la opción EDITOR y entonces aparecerán todas las opciones mencionadas. De ésta seleccionamos la opción CARGAR, aparecerá un mensaje pidiendo el nombre del archivo, entonces teclearemos TODO y daremos <ENTER>.

En caso de existir algún problema al leer el archivo se mostrará un mensaje que lo indica.



Salvar.

Esta opción sirve para salvar en un archivo, el contenido actual del editor. El nombre que MAT-UNAM asigna al archivo al arranque es NINGUND.MAT, así, cuando se desea salvar el contenido actual del editor, primero verifica si el nombre es NINGUND.MAT.

En caso de ser así, MAT-UNAM preguntará si desea salvar con este nombre o desea cambiar el nombre del archivo, para salvarlo con éste último. Para la primera alternativa se debe presionar la letra S. y para la segunda la letra C. Al dar el nombre si no se le da la extensión MAT-UNAM asumirá la extensión .MAT.

Pero en caso de que el nombre no sea NINGUND.MAT, salvará el contenido actual del editor con ese nombre, y generará adicionalmente otro archivo con extensión .BAK. Asimismo, en caso de tener algún problema al tratar de escribir el archivo nos indicará con un mensaje de error.

Error al escribir el archivo
Presione < ESC >

Salvar como.

Esta opción se utiliza para salvar el contenido actual del editor en un archivo, pero con un nombre diferente al que aparece en el encabezado. La opción asumirá que el nombre del archivo es NINGUNO.MAT, y al igual que SALVAR preguntará si desea salvar o cambiar de nombre al archivo.

Seleccione la alternativa cambiar presionando la letra C, y de el nombre que desee.

Nuevo.

Esta opción deja listo el editor para crear un nuevo archivo. Quitará todo lo que el editor contenía, asumirá el nombre del archivo como NINGUNO.MAT.

Cambiar dir.

La opción se utiliza para cambiar de unidad y el directorio de la ruta actual. En caso de existir algún problema para acceder el nuevo directorio o la nueva unidad, mostrará un mensaje de error y permanecerá en la unidad y ruta actual.

Directorio a cambiar:

Directorio.

Esta opción es utilizada para observar todos los archivos del directorio, en la unidad y ruta actual. La opción de DIRECTORIO, mostrará en una pequeña ventana, tantos archivos quepan en ésta. Cuando dicha ventana se llena se pedirá que se presione la tecla ESC para visualizar los restantes archivos.

```
C:\MATUNAM\1.*
ENTRADA.EXE  RACC.EXE    GRAFICAR.EXE  MATUNAM.EXE  TODO.MAT
FACT.MAT     LINEA.MAT   SENDO.MAT     FUNG.MAT     COMEN.MAT
COMEN.MAT    DERIVADA.MAT  VAREQUIS.MAT  MATUNAM.AYU  COMENAR.MAT
COMEN.RCC    FACT.RCC    TODO.RCC      ABCMAT.EXE   DERIVADA.RCC
                                     Presione < ESC >
```

Salir.

La opción de SALIR se utiliza para abandonar el sistema MAT-UNAM. En esta opción, se preguntará si desea salvar el contenido actual de el editor. Además, si desea abortar esta operación se presionará la tecla de ESC.

La segunda opción de la zona de menú es EVALUAR. Esta opción evaluará cada una de las operaciones que se le indiquen en la zona de ejecución. Al empezar a operar se solicitará el valor que tomarán las variables de las funciones definidas por el usuario.

También preguntará si desea que los resultados se manden a la impresora o no.

La zona de menú cuenta con un tercera opción, ésta es la denominada GRAFICAR. Esta opción se subdivide en dos opciones más que son:

+ FUNCION

+ ARCHIVO

Funcion.

La opción de FUNCION, graficará una de las funciones definidas por el usuario, actualmente en el editor. Para esto preguntará cual de las funciones definidas por el usuario es la que se graficará. En caso de ser solo una función la definida por el usuario, asumirá ésta por default.

MAT-UNAM asume que la gráfica se salvará con el nombre actual del archivo en el editor, y con extensión .RCC. pero aún así, preguntará si se desea salvar con este nombre o con otro. En caso de salvar con el nombre actual, presionar la tecla S.

En el segundo caso presionar la tecla C. y dar el nombre. Si no se da la extensión, MAT-UNAM asume la extensión .RCC.

Posteriormente, solicitará el intervalo en en cual se desea graficar. Esto es, el límite inferior y límite superior. Finalmente, preguntará si la gráfica se enviará también a la impresora.

Archivo.

La opción de ARCHIVO, graficará la función que ha sido salvada con anterioridad en un archivo con extensión RCC. También en esta opción se preguntará si se desea que la gráfica sea enviada a la impresora.

La cuarta y última opción de la zona de menú es la denominada TUTOR, esta se divide en cuatro opciones más:

- + RAIZ
- + FACTORIZAR
- + DERIVADA
- + ALGEBRA

Raiz.

Esta opción ejecuta un rutina de enseñanza de la raíz cuadrada, paso a paso, hasta obtener el resultado final. El número es dado por el usuario, en caso de ser erróneo, se mostrará un mensaje de error y regresará al editor.

Si el número es válido, entonces el usuario es guiado paso a paso, para la obtención del resultado. El usuario sólo debe presionar la tecla C.

Factorizar.

La opción de FACTORIZAR, ejecuta una rutina de enseñanza de factor común de un polinomio. De ser posible despeja la función con respecto a la variable que ha sido seleccionada.

El usuario debe proporcionar el polinomio, y la variable con respecto a la cual se desea el factor común. Si no existe ningún error de sintaxis, se indicará paso a paso la solución del problema, y el usuario sólo deberá presionar la tecla C.

En caso de existir algún error de sintaxis, se indicará de que error se trata y regresará al editor.

Derivada.

Aquí se da un método de solución de las derivadas parciales de un polinomio. El usuario debe proporcionar el polinomio, y la variable con respecto a la cual se desea la derivada parcial. Si no existe ningún error de sintaxis, se indicará paso a paso la solución del problema, y el usuario sólo deberá presionar la tecla C.

En caso de existir algún error de sintaxis, se indicará de que error se trata y regresará al editor.

Algebra.

Esta parte de tutoria, se encuentra constituida por temas básicos de algebra. Los temas que se tratan son cinco : magnitud de números, suma de números con signo, exponenciación, multiplicación y propiedad distributiva.

El usuario sólo deberá dar la respuesta correcta al problema que se plantea en el tema en cuestión. De no ser la respuesta correcta, el sistema tutor dará el resultado paso a paso.

Suma	Multiplicacion	Exponentes	P. Distributiva	Magnitud	Salir
Universidad Nacional Autonoma de México					
Facultad de Ingenieria					
SISTEMA TUTOR M A T U N A M					
<u>A L G E B R A B A S I C A</u>					

Use ← → para seleccionar y presione <ENTER>

PARTE III

Sintaxis de las funciones de MAT-UNAM.

Las funciones matemáticas válidas para MAT-UNAM son :

ABS	ATAN	CART
COS	CDT	CSC
DERIV	ENT	EXP
FACT	FRAC	INTEG
LN	LOG	POLAR
POLI	RAIZ2	SEN
SUMA	TAN	

A continuación se describirá a detalle la sintaxis de cada una de ellas.

ABS.

sintaxis : Abs(argumento)

Descripción : Valor absoluto del argumento.

donde

argumento puede ser :

- Nombre de función.
- Variable.
- Un número positivo o negativo.

Ejemplo:

```
abs(f)
abs(-456)
```

Atan.

sintaxis : Atan(argumento)

Descripción : Angulo cuyo tangente es el argumento. El argumento esta dado en grados. El resultado de la función es en grados.

donde

argumento puede ser :

- Nombre de función.
- Variable.
- Un número positivo o negativo.

Ejemplo:

```
atan(1.4)
atan(x)
```

Cart.

sintaxis : Cart(argumento1, argumento2)

Descripción : Conversión de coordenadas polares a coordenadas cartesianas. El argumento1 es el ángulo dado en grados, y el argumento2 es la magnitud.

donde

argumento1 y argumento2 pueden ser :

- Nombre de función.
- Variable.
- Un número positivo o negativo.

Ejemplo:

cart(25,10)

cart(45,x)

Cos.

sintaxis : Cos(argumento)

Descripción : Coseno del argumento. El argumento esta dado en grados. El resultado de la función es en radianes.

donde

argumento puede ser :

- Nombre de función.
- Variable.
- Un número positivo o negativo.

Ejemplo:

cos(90)

cos(g)

Cot.

sintaxis : $\text{Cot}(\text{argumento})$

Descripción : Cotangente del argumento. El argumento esta dado en grados. El resultado de la función es en radianes.

donde

argumento puede ser :

- Nombre de función.
- Variable.
- Un número positivo o negativo.

Ejemplo:

$\text{cot}(f)$

$\text{cot}(y)$

Csc.

sintaxis : $\text{Csc}(\text{argumento})$

Descripción : Cosecante del argumento. El argumento esta dado en grados. El resultado de la función es en radianes.

donde

argumento puede ser :

- Nombre de función.
- Variable.
- Un número positivo o negativo.

Ejemplo:

`csc(-89)`

`csc(x)`

Deriv.

sintaxis : `Deriv(funcion,variable)`

Descripción : Derivada parcial de la función con respecto a la variable.

donde

funcion es : Nombre de función

variable es : una variable de la función.

Ejemplo:

`deriv(f,x)`

`deriv(f,y)`

Ent.

sintaxis : `Ent(argumento)`

Descripción : Obtiene la parte entera del argumento.

donde

argumento puede ser :

- Nombre de función.
- Variable.
- Un número positivo o negativo.

Ejemplo:

ent(29.89)

ent(x)

Exp.

sintaxis : Exp(argumento)

Descripción : El número 'e' elevado al argumento.

donde

argumento puede ser :

- Nombre de función.
- Variable.
- Un número positivo o negativo.

Ejemplo:

exp(f)

exp(3)

Fact.

sintaxis : Fact(funcion,variable1[,variable2[,variable3]])

Descripción : Factor común de la función con respecto a variable1,variable2 y/o variable3. En caso de poder despejar la función con respecto a la variable1, variable2 y/o variable3 lo hará.

donde

funcion es : Nombre de función.

variable1,

variable2,

variable3 son : variables de la función.

Ejemplo:

fact(f,x)

fact(f,y,z)

Frac.

sintaxis : Frac(argumento)

Descripción : Obtiene la parte fraccionaria del argumento.

donde

argumento puede ser :

- Nombre de función.
- Variable.
- Un número positivo o negativo.

Ejemplo

frac(-10.567)

frac(x)

Integ.

sintaxis : Integ(funcion)

Descripción : Evalua la integral definida de la función.

donde

funcion es : Nombre de función

Ejemplo:

integ(f)

integ(g)

Ln.

sintaxis : Ln(argumento)

Descripción : Logaritmo natural del argumento.

donde

argumento puede ser :

- Nombre de función.
- Variable.
- Un número positivo.

Ejemplo:

ln(109)

ln(y)

Log.

sintaxis : Log(argumento)

Descripción : Logaritmo del argumento.

donde

argumento puede ser :

- Nombre de función.
- Variable.
- Un número positivo o negativo.

Ejemplo:

log(x)

log(100)

Polar.

sintaxis : Polar(variable1,variable2)

Descripción : Convierte de coordenadas cartesianas a coordenadas polares.

donde

variable1 y variable2 pueden ser :

- Nombre de función.
- Variable.
- Un número positivo o negativo.

Ejemplo:

polar(5,3)

polar(f,x)

Poli.

sintaxis : Poli(funcion,funcion1,opcion)

Descripción : Realiza la suma, resta y multiplicación de dos polinomios.

donde

funcion y funcion1 son : Nombre de función.

opcion : Es número entero entre 1 y 3.

(1 -> suma 2 -> resta 3 -> multiplicación)

Ejemplo:

poli(f,g,3)

Raiz2.

sintaxis : Raiz2(argumento)

Descripción : Obtiene la raíz cuadrada del argumento.

donde

argumento puede ser :

- Nombre de función.
- Variable.
- Un número positivo.

Ejemplo:

raiz2(y)

raiz2(36)

Sen.

sintaxis : Sen(argumento)

Descripción : Seno del argumento. El argumento esta dado en grados. El resultado de la función es en radianes.

donde

argumento puede ser :

- Nombre de función.
- Variable.
- Un número positivo o negativo.

Ejemplo:

sen(-720)

sen(z)

Suma.

sintaxis : Suma(funcion,limite1,limite2)

Descripción : Obtiene la sumatoria de la función, del limite1 al limite2.

donde

funcion es : Nombre de función.

limite1,limite2 : Un número positivo o negativo.

Ejemplo:

suma(f,2,19)

suma(g,-1,2)

Tan.

sintaxis : Tan(argumento)

Descripción : tangente del argumento. El argumento esta dado en grados. El resultado de la función es en radianes.

donde

argumento puede ser :

- Nombre de función
- Variable.
- Un número positivo o negativo.

Ejemplo:

tan(27)

tan(g)

Errores de compilación.

A continuación se mencionan los errores que se generan durante la compilación en MAT-UNAM. Se indica cual es la causa que lo provoca.

1.- Se espera nombre de la función.

Este error es debido a que se introdujo alguna función pero no se siguió correctamente la sintaxis, ya que lo primero que debe darse es el nombre que tomará la función.

2.- Se espera parentesis izquierdo.

Cuando se define la función antes de poner las variables que la función utiliza, debe colocarse el '('.

3.- Se espera variable.

Este error es debido a que no se ha dado una variable, cuando se define una función.

4.- Se espera parentesis derecho.

Cuando se define una función, al terminar de indicar las variables que la función utiliza, se debe colocar un ')'. También es debido a que se ha abierto un parentesis izquierdo y no se cerro.

5.- Se espera signo igual.

Al definir una función se debe colocar el signo '=' ,para indicar que lo que sigue es la declaración formal de la función.

6.- Se espera caracter válido.

Este error ocurre cuando se ha detectado algún símbolo raro, que no pertenece a los caracteres válidos de MAT-UNAM.

7.- Se espera número o variable.

Cuando se está declarando formalmente una función, debe darse un número o variable cuando se trata de un elemento de un término. Esto es, puede ser el coeficiente del término, o el exponente de una variable del término.

8.- Cerrando parentesis sin abrirlo.

Este tipo de error ocurre cuando no se ha abierto un '(', pero en la función aparece un ')'. Esto puede ocurrir cuando se abren y cierran muchos parentesis.

9.- Punto <.> extra en la función.

Cuando se utilizan los números de punto flotante, se puede incurrir en este tipo de error, principalmente al poner 2 puntos decimales, en lugar de uno solo.

10.- Uso inadecuado de la coma <,>.

Se debe principalmente al usar la coma en alguna función que solo contiene un argumento, y se trata de poner más de uno.

11.- Se espera punto y coma < ; >.

En la definición de una función se debe colocar al principio y fin un punto y coma. Generalmente este error ocurre cuando no se coloca un punto y coma al término de la definición de la función.

12.- Signo igual no válido en esta zona.

Se intenta hacer uso del signo igual en la zona de ejecución del editor.

13.- Nombre de función declarada 2 veces.

Este error ocurre cuando se trata de declarar una función con un nombre que ya tiene otra función.

14.- Variable declarada 2 veces.

Este error se debe a que una variable ya se ha declarado y se intenta volver a declarar.

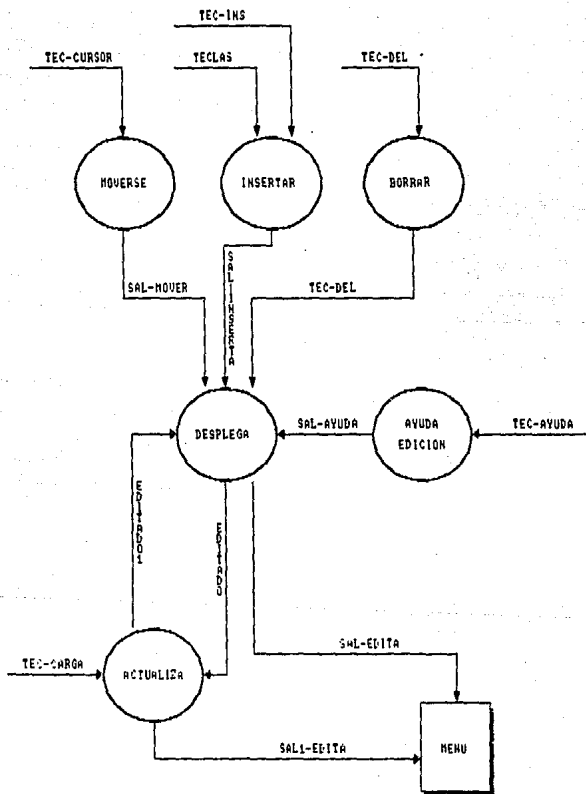
15.- Operador inválido al inicio de función.

En MAT-UNAM no puede existir un operador de multiplicación o división (*, /) al inicio de una función. Los operadores validos son de suma y resta (+, -).

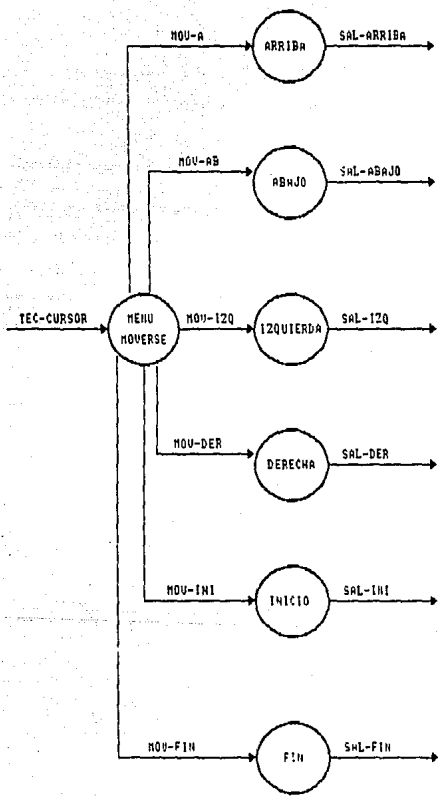
16.- Variable o función no declarada.

Se debe cuando se trata de utilizar el nombre de una función, o una variable que no existe.

APENDICE B
DIAGRAMA DE FLUJO DE DATOS
DEL SISTEMA MAT-UNAM

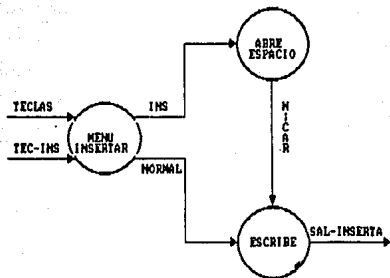


Modulo de Edicion



Modulo Edicion Moverse

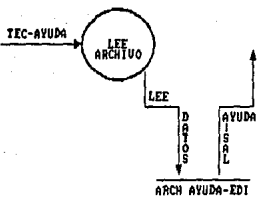
Modulo Edicion Insertar.



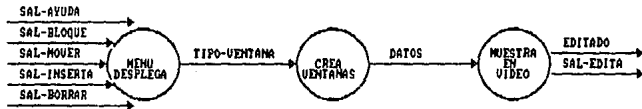
Modulo Edicion Borrar.



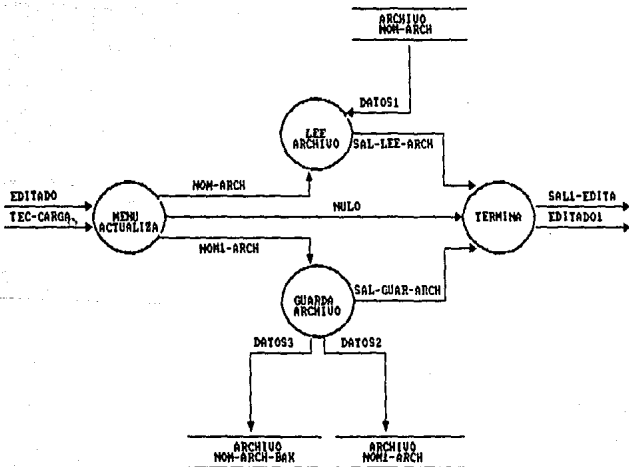
Modulo Edicion Ayuda.

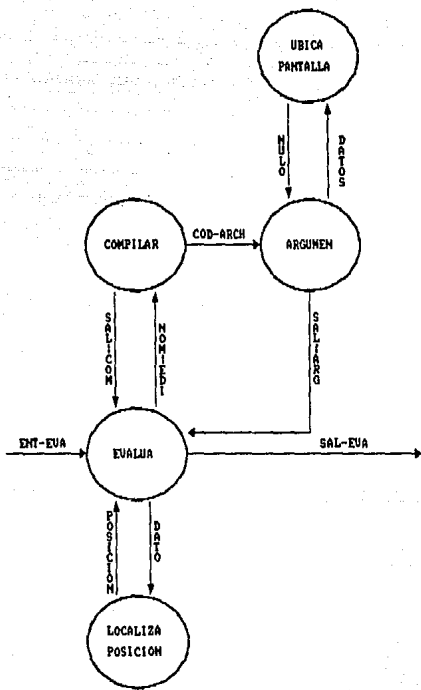


Modulo Edicion Desplega.

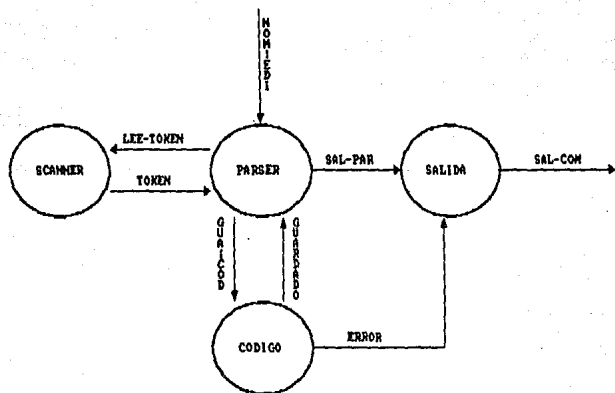


Modulo Edicion Actualiza.

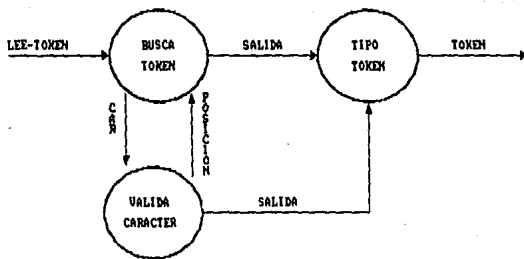




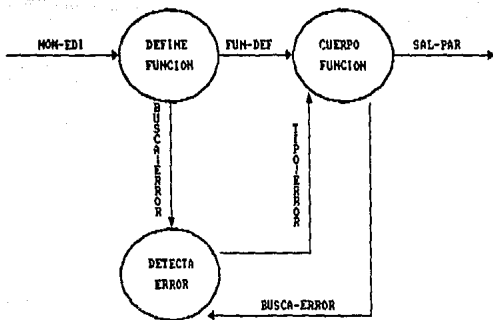
Modulo Evaluar.



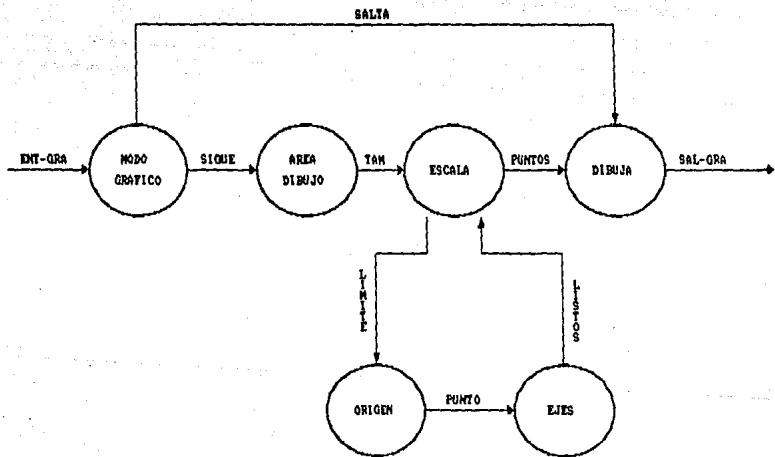
Modulo Evaluar Compiler.



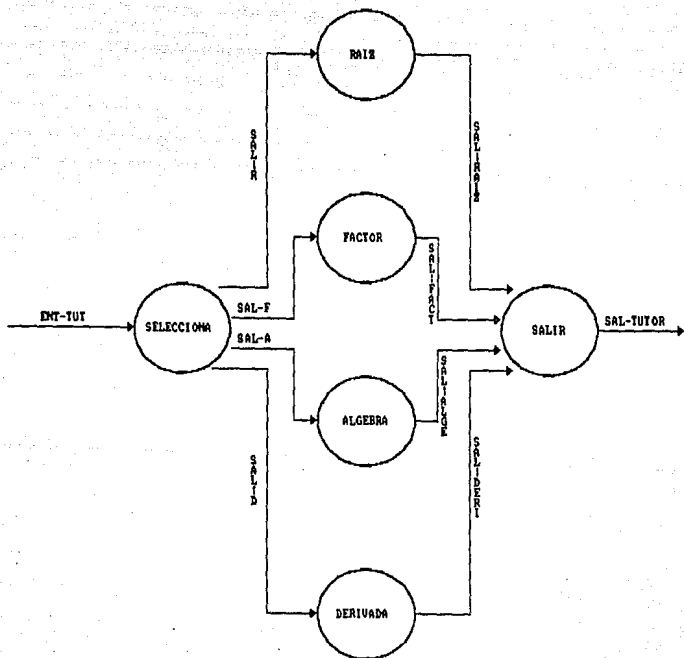
Modulo Evaluar Scanner.



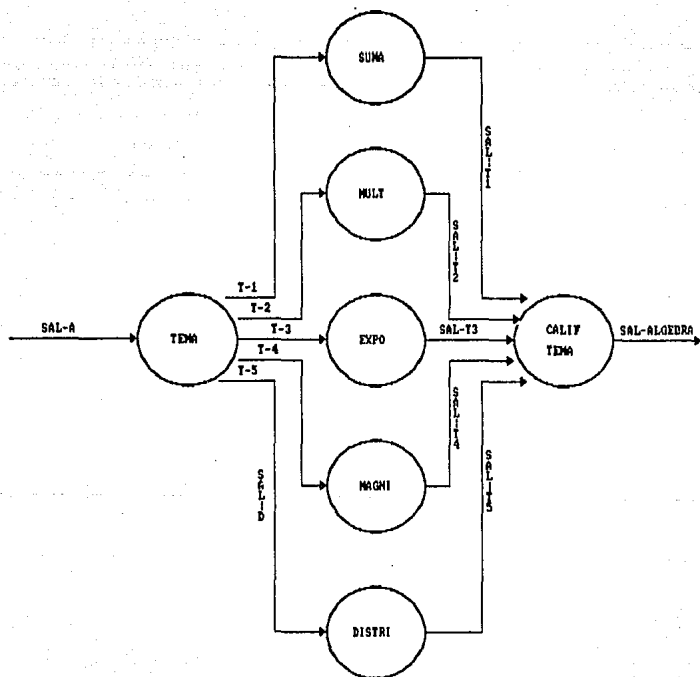
Modulo Evaluar Parser.



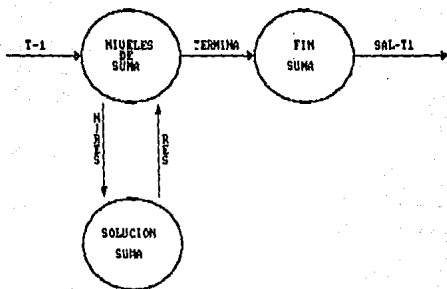
Modulo Graficar.



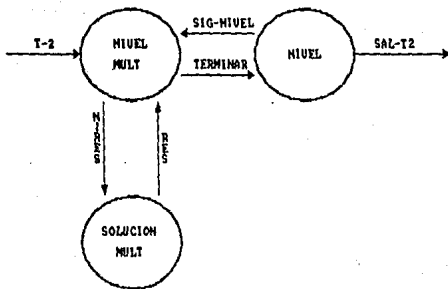
Modulo Tutor.



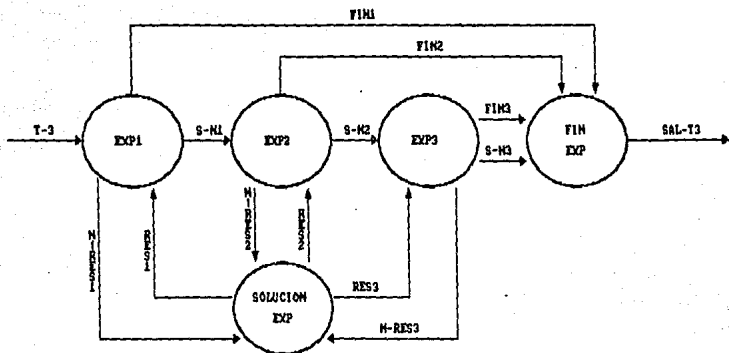
Modulo Tutor Algebra.



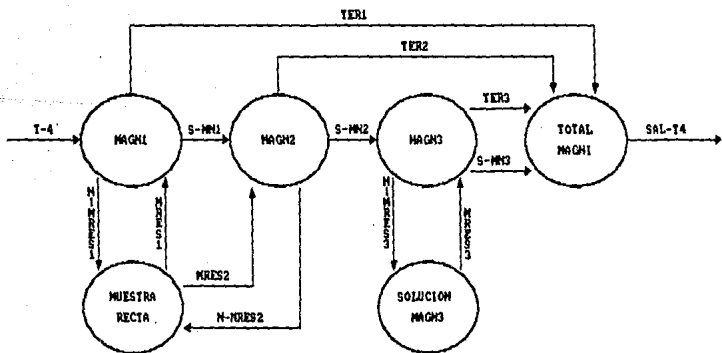
Modulo Tutor Suma.



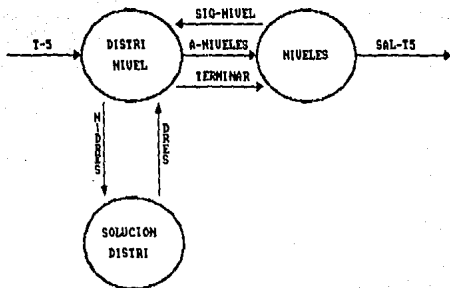
Modulo Tutor Mult



Modulo Tutor Expo.



Modulo Tutor Magni.

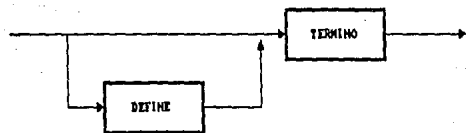


Modulo Tutor Distri.

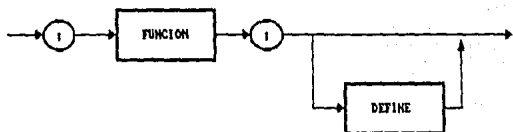
APENDICE C
GRAMATICA DEL COMPILADOR
DE MAT-UNAM

Gramatica de funciones matematicas.

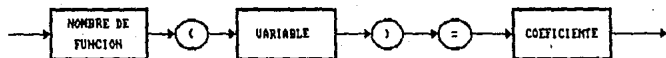
GRAMAFUN:



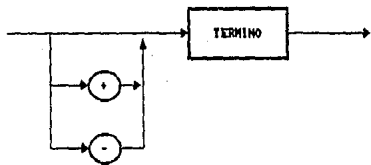
DEFINE:



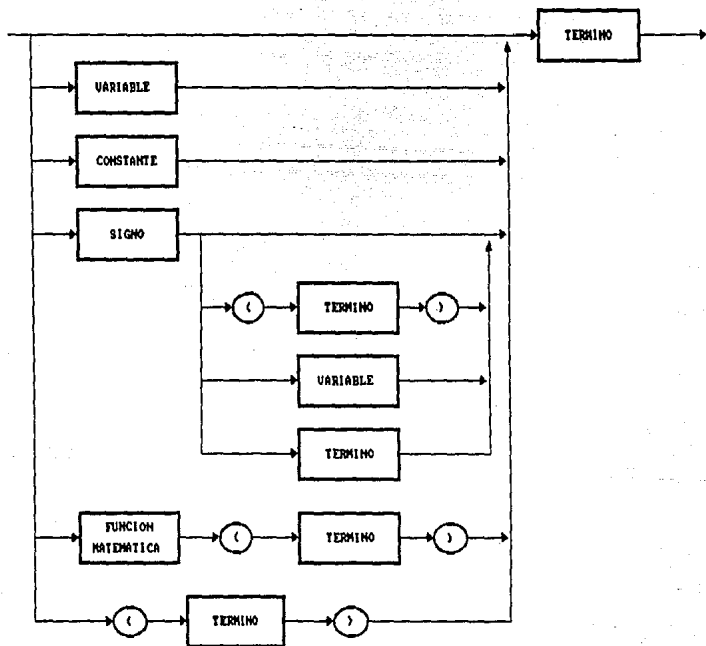
FUNCION:



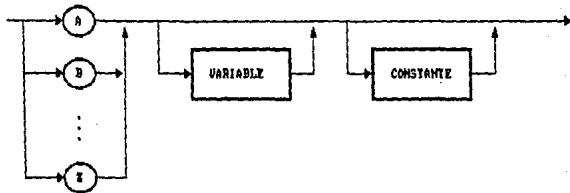
COEFICIENTE:



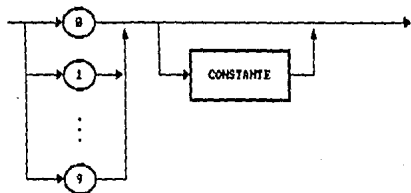
TERMINO:



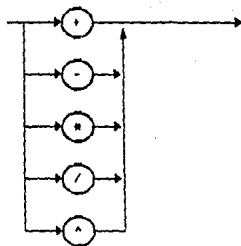
VARIABLE:



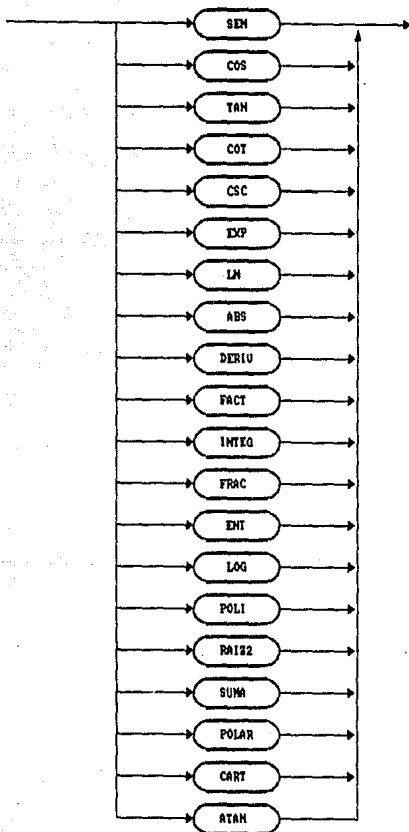
CONSTANTE:



SIGNO:



FUNCIÓN MATEMÁTICA:



APENDICE D

OPERACIONES CON POLINOMIOS

El presente apéndice se divide en dos secciones fundamentales de operación del sistema MAT-UNAM. La primera de ellas se denomina errores comunes, en la cuál se muestran algunos de los errores que con mayor frecuencia se presentan en el estudiante de matemáticas.

En esta sección se muestran unas pantallas, las cuales están divididas en dos partes, el lado izquierdo muestra la operación con el error y el lado derecho la misma operación en forma correcta. Además, en la parte superior de la pantalla se indica el tema de que se trata.

La segunda sección se denomina manejo de polinomios. En dicha sección, se presentan las operaciones que pueden realizarse con los polinomios en el sistema MAT-UNAM. Al igual que en la sección anterior se muestran las pantallas correspondientes.

I.- ERRORES COMUNES

Elegir opcion	Salir
<p>Universidad Nacional Autonoma de México Facultad de Ingenieria SISTEMA TUTOR M A T U N A M</p> <p><u>ERRORES COMUNES</u></p>	

Use <- -> para seleccionar y presione <ENTER>

Elegir opcion	Salir									
<table border="1"><tr><td>Desigualdades</td></tr><tr><td>Radicales</td></tr><tr><td>Exponentes</td></tr><tr><td>Factorizacion</td></tr><tr><td>Fracciones</td></tr><tr><td>Ec. linea</td></tr><tr><td>Logaritmos</td></tr><tr><td>Trigonometria</td></tr></table>	Desigualdades	Radicales	Exponentes	Factorizacion	Fracciones	Ec. linea	Logaritmos	Trigonometria	<table border="1"><tr><td style="text-align: center;"><p>Universidad Nacional Autonoma de México Facultad de Ingenieria SISTEMA TUTOR M A T U N A M</p><p><u>ERRORES COMUNES</u></p></td></tr></table>	<p>Universidad Nacional Autonoma de México Facultad de Ingenieria SISTEMA TUTOR M A T U N A M</p> <p><u>ERRORES COMUNES</u></p>
Desigualdades										
Radicales										
Exponentes										
Factorizacion										
Fracciones										
Ec. linea										
Logaritmos										
Trigonometria										
<p>Universidad Nacional Autonoma de México Facultad de Ingenieria SISTEMA TUTOR M A T U N A M</p> <p><u>ERRORES COMUNES</u></p>										

Use <- -> para seleccionar y presione <ENTER>

Errores comunes de : DESIGUALDADES

INCORRECTO

$$| 3/4 - 2 | = 3/4 - 2 = -5/4$$

$$| 5 - 7 | = | 5 | - | 7 | = -2$$

$| x | = -2$
tiene la solución $x = -2$

$$| x-1 | < 3 \text{ si y solo si } x < 4$$

CORRECTO

$$| 3/4 - 2 | = | -5/4 | = 5/4$$

$$| 5 - 7 | = | -2 | = 2$$

No hay solución; el valor absoluto de un número nunca puede ser negativo

$$| x-1 | < 3 \text{ si y solo si } -3 < x-1 < 3$$

esto es $-2 < x < 4$

Presione << C >> para continuar

SISTEMA MAT-UNAM

Errores comunes de : RADICALES

INCORRECTO	CORRECTO
$\sqrt{9+16} = \sqrt{9} + \sqrt{16}$ $\sqrt[3]{8} \cdot \sqrt[2]{8} = \sqrt[6]{64}$ $\sqrt{(x-1)^2} = x-1$ $\sqrt[9]{x} = x^3$	$\sqrt{9+16} = \sqrt{25}$ $\sqrt[3]{8} \cdot \sqrt[2]{8} = 2 \cdot 2 \cdot \sqrt{2} = 4 \cdot \sqrt{2}$ $\sqrt{(x-1)^2} = x-1 $ $\sqrt[9]{x} = \sqrt[3]{x} \cdot \sqrt[3]{x} = x \cdot \sqrt[3]{x}$
<p>Presione << C >> para continuar</p>	

SISTEMA MAT-UNAM

Errores comunes de : EXPONENTES

INCORRECTO	CORRECTO
$5^2 \cdot 5^4 = 5^8$ (No multiplican los exponentes)	
$5^2 \cdot 5^4 = 25^6$ (No multiplican las bases)	$5^2 \cdot 5^4 = 5^6$ (Regla 1)
$5^6 / 5^2 = 5^3$ (No divide los exponentes)	
$5^6 / 5^2 = 1^4$ (No divide las bases)	$5^6 / 5^2 = 5^4$ (Regla 2)

Presione << C >> para continuar

SISTEMA MAT-UNAM

Errores comunes de : EXPONENTES

INCORRECTO	CORRECTO
$[5^2]^6 = 5^8$ (No suma los exponentes)	$[5^2]^6 = 5^{12}$ (Regla 3)
$(-2)^4 = -2^4$ (No ignore los parentesis)	$(-2)^4 = (-1)^4 \cdot 2^4 = 2^4$ (Regla 4)
$(-5)^0 = -1$ (No ignore la definicion de b)	$(-5)^0 = 1$ (Definicion de b)
$2^{-3} = -1 / 2^3$ (No ignore la definicion de b ⁻ⁿ)	$2^{-3} = 1 / 2^3$ (Definicion de b ⁻ⁿ)

Presione << C >> para continuar

SISTEMA MAT-UNAM

D-6

Errores comunes de : FACTORIZACION

INCORRECTO	CORRECTO
$(x + 2)^3 + (x + 2)y - (x + 2)^3y$	$(x + 2)^3 + (x + 2)y - (x + 2)(3 + y)$
$3x + 1 = 3(x + 1)$	$3x + 1$ No es factorizable usando enteros
$x^3 + 8$ No es factorizable.	$x^3 + 8 = (x + 2)(x^2 - 2x + 4)$
$x^2 + y^2 = (x + y)(x + y)$	$x^2 + y^2$ No es factorizable usando numeros reales.
Presione << C >> para continuar	

SISTEMA MAT-UNAM

Errores comunes de : OPERACIONES CON FRACCIONES

INCORRECTO	CORRECTO
$\frac{1}{a} + \frac{1}{b} = \frac{1}{a+b}$	$\frac{1}{a} + \frac{1}{b} = \frac{b+a}{ab}$
$\frac{2}{3} + \frac{x}{5} = \frac{2+x}{3+5}$	$\frac{2}{3} + \frac{x}{5} = \frac{2 \cdot 5 + 3 \cdot x}{3 \cdot 5} = \frac{10 + 3x}{15}$
$3 \left[\frac{x+1}{x-1} \right] = \frac{3(x+1)}{3(x-1)}$	$3 \left[\frac{x+1}{x-1} \right] = \frac{3(x+1)}{x-1}$
<p>Presione << C >> para continuar</p>	

SISTEMA MAT-UNAM

Errores comunes de : ECUACION DE LA LINEA

INCORRECTO	CORRECTO
<p>La pendiente de la línea entre (2,3) y (5,7) es:</p> $m = \frac{7 - 3}{2 - 5} = -\frac{4}{3}$ <p>La pendiente de la línea $2x - 3y = 7$ es $-2/3$</p> <p>La línea entre $(-4, -3)$ con pendiente 2 es: $y - 3 = 2(x - 4)$</p>	<p>La pendiente de la línea entre (2,3) y (5,7) es:</p> $m = \frac{7 - 3}{2 - 5} = \frac{4}{3}$ <p>La pendiente es $2/3$</p> <p>La ecuación es: $y - (-3) = 2(x - (-4))$ o bien $y + 3 = 2(x + 4)$</p>
<p>Presione << C >> para continuar</p>	

SISTEMA MAT-UNAM

Errores comunes de : LOGARITMOS

INCORRECTO	CORRECTO
$\log(A) + \log(B) = \log(A + B)$	$\log(A) + \log(B) = \log(AB)$
$(\log x)^2 = 2 \cdot \log(x)$	$(\log x)^2 = (\log(x))(\log(x))$
$\log(A) - \log(B) = \log(A) / \log(B)$	$\log(A) - \log(B) = \log(A/B)$
$\log(x/2) = \log(x)/2$	$\log(x/2) = \log(x) - \log(2)$
$\log(x + 2)^2 = 2 \cdot \log(x + 2)$	$\log(x + 2)^2 =$ No puede ser simplificado aun mas.
Presione << C >> para continuar	

SISTEMA MAT-UNAM

Errores comunes de : TRIGONOMETRIA

INCORRECTO	CORRECTO
$\cos(-50) = -\cos(50)$ $\cos(125) = \cos(55)$ $\operatorname{sen}\left[-\frac{\pi}{3}\right] = \frac{\sqrt{3}}{2}$ $\operatorname{sec}\frac{\pi}{2} = \frac{1}{\cos\frac{\pi}{2}}$	$\cos(-50) = \cos(50)$ $\cos(125) = -\cos(55)$ $\operatorname{sen}\left[-\frac{\pi}{3}\right] = -\operatorname{sen}\frac{\pi}{3} = -\frac{\sqrt{3}}{2}$ $\cos\frac{\pi}{2} = 0; \operatorname{sec}\frac{\pi}{2} \text{ No esta definido}$
Presione << C >> para continuar	

SISTEMA MAT-UNAM

Errores comunes de : TRIGONOMETRIA

INCORRECTO	CORRECTO
$\cos^4(x) = \frac{1 + \cos^2(2x)}{2}$ $\operatorname{sen}(4x) = 4 \cdot \operatorname{sen}(x)$	$\cos^4(x) = \left[\frac{1 + \cos(2x)}{2} \right]^2$ $\operatorname{sen}(4x) = \operatorname{sen}(2(2x))$ $= 2 \cdot \operatorname{sen}(2x) \cdot \cos(2x)$
Presione << C >> para continuar	

SISTEMA MAT-UNAM

II. - MANEJO DE POLINOMIOS

Editor Evaluar Graficar Tutor
D:\MATUNAM\SUMA.MAT

```
:f(x,y,z) = x^2*y + 5*x*y - 3*x*y^2 ;  
:g(x,y,z) = 2*x^2*y - 5*x*y^2 + 3*x*y;  
poli(f,g,1)
```

SISTEMA-TUTOR Ayuda F1 Menu F10 Salir ALT-X R:1 C:1 R A C C

Evaluando

presione < ESC >

```
X^2Y+5XY-3XY^2  
2X^2Y-5XY^2+3XY  
+3X^2Y+8XY-8XY^2
```

Editor Evaluar Graficar Tutor
D:\MATUNAM\RESTA.MAT

```
f(x,y,z) = x^2*y + 5*x*y - 3*x*y^2 ;  
g(x,y,z) = 2*x^2*y - 5*x*y^2 + 3*x*y;  
poli(f,g,2)
```

SISTEMA-TUTOR Ayuda F1 Menu F10 Salir ALT-X R:1 C:1 R A C C

E v a l u a n d o

presione < ESC >

```
X^2Y+5XY-3XY^2  
2X^2Y-5XY^2+3XY  
-1X^2Y+2XY+2XY^2
```

Editor Evaluar Graficar Tutor
D:\MATUNAM\MULTIPLI.MAT

```
;f(x,y,z) = x^2*y + 5*x*y -3*x*y^2 ;  
;g(x,y,z) = 2*x^2*y ;  
poli(f,g,3)
```

SISTEMA-TUTOR Ayuda F1 Menu F10 Salir ALT-X R:2 C:22 R A C C

Evaluando

presione < ESC >

```
X^2Y+5XY-3XY^2  
2X^2Y  
+2X^4Y^2+10X^3Y^2-6X^3Y^3
```


Editor Evaluar Graficar Tutor
D:\MATUNAM\FACT.MAT

```
; f ( x, y, z ) = x^2 + 2*x*y^2 - 4*x^2*z + 5*z*y ;  
fact(f,x)
```

SISTEMA-TUTOR Ayuda F1 Menu F10 Salir ALT-X R:1 C:1 R A C O

E v a l u a n d o

presione < ESC >

```
X ( X+2Y^2-4XZ ) +5YZ - 0  
Expresion minima
```

Editor

Evaluar

Graficar

Tutor

D:\MATUNAM\LINEA.MAT

```
: f( x,y) = 5*x+2-6*y +3*x -2*x -3 ;
```

```
fact(f,x)
```

SISTEMA-TUTOR Ayuda F1 Menu F10 Salir ALT-X R:1 C:1 R A C C

Evaluando

presione < ESC >

```
X = (+1+6Y) / 6
```

Editor

Evaluar

Graficar

Tutor

D:\MATUNAM\DERIVADA.MAT

: f(x,y,z) = x*y*z - y^2*z - 2*x^3 + 3*x*z^2 + 1 ;

deriv(f,x)

deriv(f,y)

deriv(f,z)

SISTEMA-TUTOR Ayuda F1 Menu F10 Salir ALT-X R:1 C:1 R A C C

Evaluando

deriv(F,Z)

presione < ESC >

deriv(F,X)

$$\frac{dF}{dX} = YZ - 6X^2 + 3Z^2$$

deriv(F,Y)

$$\frac{dF}{dY} = XZ - 2YZ$$

deriv(F,Z)

$$\frac{dF}{dZ} = XY - 1Y^2 + 6XZ$$