Instituto Politécnico Nacional
Escuela Superior de Cómputo
Departamento de Programación y Desarrollo de Sistemas

Desarrollo Didáctico para la Unidad de Recursión

Diplomado: FORMACIÓN Y ACTUALIZACIÓN PARA UN NUEVO MODELO EDUCATIVO

Araujo Díaz David

México, D.F. marzo del año 2005

Contenido

Página

Módulo de Desarrollo Didáctico
Actividad Integradora

"Desarrollo Didáctico para la Unidad de Recursión"

David Araujo Díaz

1. Objetivos Generales de este Trabajo

Hay dos objetivos generales:

- Que los participantes se apropien de los elementos conceptuales y metodológicos relacionados con la **Planeación Docente** y el **Diseño Didáctico**.

- Integrar conocimientos aportados por el **Diplomado**, mediante el diseño de una **Unidad Didáctica** centrada en el aprendizaje, acorde al contexto del participante.

2. Actividad Integradora

Una **Unidad Didáctica** es una estrategia de enseñanza aprendizaje, en la que se precisan sus objetivos particulares y generales, se definen acciones específicas, se indican magnitudes y secuencias propias de los objetivos y se determinan los medios y recursos apropiados, no sólo relacionados con el tema, sino también respecto al aprendizaje.

A continuación se describe la unidad didáctica relacionada con la asignatura de **Programación III**, impartida en el **tercer semestre** de la carrera de **Ingeniería en Sistemas Computacionales** en la **Escuela Superior de Cómputo** (**ESCOM-IPN**).

2.1 Presentación

La asignatura de **Programación III** forma parte de la carrera de **Ingeniería en Sistemas Computacionales** de la **Escuela Superior de Cómputo** (**ESCOM-IPN**). Se imparte en el **tercer** semestre, de dicha carrera.

2.2 Justificación

Se eligió para esta presentación la unidad de **recursión**, por ser un tema que ilustra la forma en que los alumnos trabajarán el resto del semestre, que permite conocer los conocimientos de programación de los estudiantes, que se presta para que los estudiantes demuestren sus habilidades de programación, colaboración y análisis en la resolución de problemas, que les permite conocer nuevas tecnologías y que fomenta la creatividad de cada uno de ellos.

Permite además el empleo de diversas herramientas y procedimientos que facilitan el aprendizaje de los estudiantes y que fomentan el interés por la asignatura y por la carrera; como veremos durante el desarrollo de este documento. Esta parte de la unidad didáctica se diseño en base a los temas del **Programa Oficial**, para la asignatura de **Programación III**, modificándolos desde un punto de vista constructivista. Los datos de la asignatura, se muestran a continuación:

Escuela:	**Escuela Superior de Cómputo**
Departamento:	**Departamento de Programación y Desarrollo de Sistemas**
Carrera:	**Ingeniería en Sistemas Computacionales**
Clave de la Asignatura:	**COMPU302**
Semestre:	**Tercero**
Teoría:	**63.0 Horas**
Laboratorio:	**18.0 Horas**

2.3 Unidad Didáctica

"Desarrollo Didáctico para la Unidad de Recursión"

2.3.1 Presentación de la Unidad Didáctica

Esta unidad forma parte de la asignatura de **Programación III** (**Estructuras de Datos**), como se muestra en la **Figura 1**.

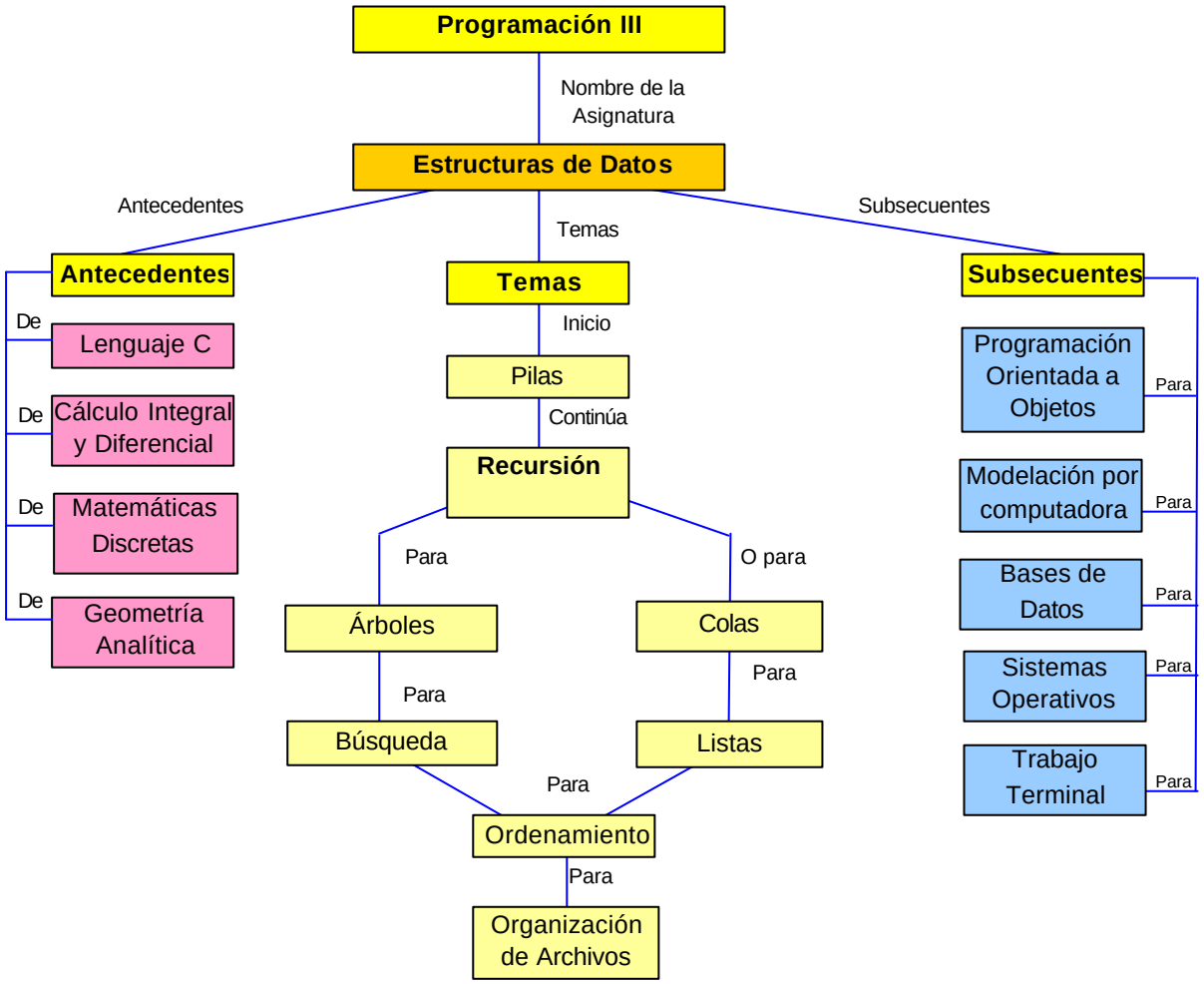


Figura 1: Mapa conceptual en donde se ubica la Unidad Didáctica.

Los conocimientos específicos sobre otras áreas, vinculados a los contenidos que se desarrollarán, son:

- **Programación**.- conocimiento de programación con **lenguaje C**.
- **Cálculo**.- calculo diferencial e integral y geometría analítica.
- **Matemáticas discretas** - convergencia de funciones matemáticas.

2.3.2 Objetivos

Objetivo de Asignatura

- El alumno analizará las estructuras de datos con las que se logra un uso óptimo de los recursos de la computadora, y aplicará las técnicas de programación para escribir programas útiles y portables, que implementen y manejen estas estructuras.

Objetivos Particulares por Unidad

Se pretende que alumno cumpla los siguientes objetivos por tema:

- **Pilas**.- El alumno analizará la estructura de datos tipo pila y los aplicará para resolver problemas de programación, y los relacionará con problemas de la vida cotidiana.
- **Recursión**.- El alumno aplicará los conceptos de funciones recursivas para resolver problemas de programación, y observará que no todos los problemas se pueden resolver en un tiempo razonable.
- **Colas**.- El alumno aplicará colas para modelar y resolver problemas y como herramientas en la elaboración de sistemas más grandes.
- **Listas**- El alumno aplicará listas para modelar y resolver problemas y como herramientas en la elaboración de sistemas de información complejos.
- **Árboles**.- El alumno aplicará diversas formas de árboles para modelar problemas, diseñar algoritmos e implementarlos, además conocerá algoritmos de compresión de información.
- **Ordenamiento**.- El alumno conocerá y analizará los algoritmos más utilizados para el ordenamiento, implementando los algoritmos de forma práctica. Cada algoritmo se analizará desde el punto de vista de la complejidad computacional.
- **Búsqueda**.- El alumno conocerá y analizará los algoritmos más utilizados para la búsqueda, implementando los algoritmos de forma práctica.
- **Organización de Archivos**.- El alumno aplicará los principales métodos para manejo de información en sistemas de almacenamiento secundario y determinará el algoritmo a utilizar en cada caso.

2.3.3 Procedimientos de enseñanza-aprendizaje

Metodología a emplear

- Metodología de aprendizaje grupal desarrollando aplicaciones prácticas, con participación activa y búsqueda de información que facilite el entendimiento de los aspectos teóricos, análisis y solución de algoritmos.

Habilidades a desarrollar en los alumnos

- Crear alumnos competentes para aplicar matemáticas, algoritmia y programación para solucionar problemas relacionados con el área de la computación y demás tecnologías afines.
- Fomentar la investigación en el área de cómputo, para el análisis de algoritmos complejos.

Valores y actitudes a promover en los alumnos

- Actitud positiva hacia el trabajo en equipo.
- Valores de respeto hacia el trabajo de sus compañeros.
- Los valores y actitudes a desarrollar durante el curso: diálogo, tolerancia, alegría y solidaridad.

2.3.4 Materiales y recursos de apoyo

Instalaciones y Medios Didácticos

A continuación se muestra un listado de los medios disponibles y sus condiciones, obra editorial, biblioteca, salones, laboratorios, centros de documentación, equipo audiovisual, recursos computacionales y de comunicaciones.

- **Recursos generales**.- pizarrón, plumones, borrador, etc.
- **Medios**.- Acetatos de los apuntes de materia disponibles en la página de Internet:

  `www.geocities.com/daraujo18`

- **Proyector multimedia**.- para proyectar y explicar el funcionamiento de los programas del curso.
- **Computadora**.- para realizar las prácticas de laboratorio.
- **Internet**.- consultar otras fuentes sobre la matera de métodos numéricos, además de las páginas del curso:

  `www.geocities.com/daraujo18` y `www.geocities.com/daraujo20`

- **Laboratorio de cómputo**.- para trabajar con las computadoras, será necesario que cada una cuente con el **lenguaje C** y el sistema operativo necesario para su puesta en funcionamiento.
- **Obra editorial**.- se aconseja que los alumnos consulten las siguientes referencias, además algunos de estos libros se encuentran en la biblioteca del plantel, y todos se encuentran en la **Biblioteca Nacional de Ciencia y Tecnología**.
- **Observatorio de Visualización de la UNAM**.- Herramienta para visualizar imágenes en tercera dimensión, ideal para mostrar ejemplos de cómo se combinan la **electrónica** y la **computación** para crear un espacio de encuentro multidisciplinarlo, y generar imágenes y percepciones impresionantes, capaces de motivar el interés de los estudiantes por el tema, por su carrera y por adquirir conocimientos en otras áreas (**Apéndice C**).

Referencias Bibliográficas

- **[Burden 02]** Burden, Richard L. y Faires, J. Douglas; "**Análisis Numérico**", 7ma ed., Ed. Thomson-Learning, México, 2002,839pp.

- **[Chapra 96]** Chapra, Steven C. y Canale, Raymond P.; "**Métodos Numéricos para Ingenieros, con aplicaciones en computadoras personales**", Ed. McGraw-Hill, México, 1996, 641pp.

- **[Esakov 89]** Esakov, J. y Weiss, T.; "**Data Structures. An Advanced Approach Using C**", Ed. Prentice Hall, U.S.A., 1989, 372pp.

- **[Heileman 98]** Heileman, Gregory L.; "**Estructuras de Datos, Algoritmos y Programación Orientada a Objetos**", Ed. McGraw-Hill, 1998, 305pp.

- **[Langsam 97]** Langsam, Y., Augenstein, M. y Tenenbaum, A.; "**Estructuras de Datos con C y C++**", 2ª ed., Ed. Prentice Hall, México, 1997, 672pp.

- **[Nakamura 92]** Nakamura, Shoichiro; "**Métodos Numéricos Aplicados con Software**", Ed. Prentice Hall, México, 1992, 570pp.

- **[Nieves 99]** Nieves, A. y Domínguez, Federico C.; "**Métodos Numéricos Aplicados a la Ingeniería**", Ed. CECSA, México, 1999, 602pp.

2.3.5 Planeación en el aula

A continuación se muestra la forma en que será abordará la unidad en el aula, donde se desglosan los objetivos específicos, el tiempo por cada tema, las actividades, los recursos y la forma evaluación de la **Unidad Didáctica** descrita. Los temas tratados seguirán los apuntes escritos para este curso (**Apéndice A**) y los aprendizajes podrán ser evaluados por los alumnos (**Apéndice B**).

Unidad 2: Recursión (3.0 horas)

- **Objetivo:** El alumno aplicará los conceptos de funciones recursivas para resolver problemas de programación, y observará que no todos los problemas se pueden resolver en un tiempo razonable.

Tema	Objetivos específicos	Horas	Actividades	Recursos	Evaluación
2	**Recursión.-** El alumno aplicará los conceptos de funciones recursivas para resolver problemas de programación, y observará que no todos los problemas se pueden resolver en un tiempo razonable	**3.0**	Exposición del profesor (**Apéndice A**), participación e investigación de aplicaciones por parte de los estudiantes.	Pizarrón, lecturas adicionales, acetatos y visita a la **sala iXtli** (**Apéndice C**).	Tareas (**Apéndice B**), trabajo de investigación, examen departamental y desarrollo de programas para la **sala iXtli**.
2.1	**Recursión.-** el alumno analizará y comprenderá el concepto y la definición de las funciones recursivas	1.5	Exposición del profesor (**Apéndice A**), participación e investigación de aplicaciones por parte de los estudiantes.	Pizarrón, lecturas adicionales y acetatos.	Tareas (**Apéndice B**) y trabajo de investigación.
2.2	**Ejemplos de recursión.-** el alumno implantará y analizará diferentes ejemplos de funciones recursivas, desde el punto de vista de complejidad computacional	1.5	Exposición del profesor (**Apéndice A**) y participación por parte de los estudiantes. Preparación de una clase en la **sala iXtli** (**Apéndice C**)	Pizarrón, lecturas adicionales, acetatos y visita a iXtli (**Apéndice C**).	Tareas (**Apéndice B**), trabajo de investigación y desarrollo de programas para la **sala iXtli** (**Apéndice C**).

2.3.6 Procedimientos de evaluación

Para determinar en qué medida se logran los objetivos generales y específicos, así como para juzgar el aprovechamiento del alumno y, para formular juicios respecto al profesor, los métodos, y los medios empleados, se describen los siguientes puntos.

Modelo y técnicas de evaluación

Se empleo el **modelo constructivista**, para llevar a los estudiantes a que aprendan por si mismos, la evaluación se da en diferentes momentos, desde el inicio, es decir, la **evaluación diagnóstica** que sirve para identificar los conocimientos previos, la **evaluación formativa** o **continua** que se da en el desarrollo y cuyo objetivo es retroalimentar e ir viendo los avances y la **evaluación sumativa** que se da al final y sirve para asignar una calificación considerando todas las anteriores.

Estrategias y procedimientos de evaluación

Se evaluará a los estudiantes con:

- Tres exámenes departamentales.
- Doce prácticas de laboratorio.
- Tareas, participaciones y visitas realizadas a lo largo del semestre.

Criterios para calificar a los estudiantes

Los estudiantes serán evaluados en los tiempos marcados por el calendario oficial, que permitan la evaluación departamental como se muestra en la siguiente tabla.

Período	Unidad	Procedimiento de evaluación
*	*	Examen diagnóstico sin valor
1	I, II, III	Examen escrito **60%** + prácticas **20%** + tareas **20%** = **100 %**
2	IV, V, VI	Examen escrito **60%** + prácticas **20%** + tareas **20%** = **100 %**
3	VII, VIII	Examen escrito **60%** + prácticas **20%** + tareas **20%** = **100 %**

Nota: en las tareas se encentran consideradas las participaciones en clase en un **10%** y en las prácticas se encuentran consideradas visitas a lugares de interés (como la sala **iXtli**) en un **5%**.

Evaluación de resultados

En base a los diagnósticos realizados al inicio del semestre y con los resultados obtenidos al final, se evaluará el proceso de planeación, tomando en cuenta los siguientes aspectos:

- **Desempeño inicial y final del grupo**; los grupos atendidos con la planeación propuesta se deberán de mantener motivados, esperando un **90%** de asistencia, y se deberá cubrir el **100%** de los temas del programa, además de cumplir con los tiempos establecidos.

- Nivel de participación del grupo en actividades como investigación, trabajo en equipo, asesorías, etc. tomando la cantidad de cada una de estas actividades al inicio y al final del semestre.

3. Conclusiones

La labor docente no es sencilla, requiere de un gran esfuerzo, no basta sólo con tener los conocimientos teóricos y técnicos sobre un área especifica, hace falta además conocer la forma de involucrar a los estudiantes en el proceso de su propio aprendizaje. Esta complejidad de la práctica docente se refleja en el **proceso de planeación** de una **unidad didáctica**, usada como una estrategia de enseñanza aprendizaje, en la que se define y específica todo lo relacionado con un tema de una asignatura.

Durante el desarrollo de la **unidad didáctica** para el tema de la **recursión**, que forma parte de la asignatura de **Programación III** (estructuras de datos), de la **ingeniería en sistemas computacionales** de la **ESCOM**, se definieron los objetivos, ajustándolos al modelo constructivista, en donde los estudiantes hagan suyos los aprendizajes, y en donde no sólo el profesor sea el responsable del aprendizaje de los alumnos, son ellos los encargados de su propio aprendizaje.

Se definieron algunas acciones específicas, como la forma en que se llevará a cabo una clase, la cuál consiste en una breve exposición por parte del profesor, la participación de los estudiantes discutiendo los resultados de sus investigaciones, y en resolver algunas dudas, para después contestar las preguntas de auto evaluación, las cuales tienen el propósito de que los alumnos conozcan cuanto han comprendido y aprendido del tema correspondiente.

Se determinaron además una serie de secuencias apropiadas a los objetivos, para que los alumnos construyan sus propios conceptos y se apropien de los procedimientos en la resolución de problemas reales, así pues al final de dicha unidad se propone a los estudiantes trabajar en la **Sala iXtli**, realizando algún proyecto relacionado con la unidad didáctica; con ello se pretende que los alumnos se relacionen y trabajen de forma colaborativa con estudiantes de otras instituciones educativas, que se interesen por conocer como trabajan equipos de cómputo, electrónica y comunicaciones avanzados, que comprendan la forma en que trabajan los equipos de súper cómputo, etc.

La **evaluación** debe incluir actividades de estimación **cualitativa** o **cuantitativa**, las cuales se deben de considerar imprescindibles, pero al mismo tiempo involucra otros factores que van más allá y que en cierto modo la definen, estos son:

1. **Demarcación del objeto**, situación o nivel de referencia que se ha de evaluar; por ejemplo, la identificación de los objetos de evaluación.

2. **Uso de determinados criterios** para la realización de la evaluación. Existen dos tipos de criterios: de **realización** (nombran los actos concretos que se esperan de los alumnos) y de **resultados** (contemplan aspectos tales como: pertinencia, precisión, originalidad, volumen de conocimientos utilizados, etc.).

3. Cierta **sistematización** mínima necesaria para la obtención de la información. La sistematización se consigue mediante la aplicación de las diversas técnicas, procedimientos e instrumentos evaluativos que hagan emerger los indicadores en el objeto de evaluación, según sea el caso y su pertinencia.

4. A partir de la **obtención de la información** y mediante la **aplicación de las técnicas** será posible construir una representación lo más fidedigna posible del objeto de evaluación.

5. **Emisión de juicios**. Con base en los puntos anteriores será posible elaborar un juicio de naturaleza esencialmente cualitativa sobre lo que hemos evaluado. La elaboración del juicio nos permite realizar una interpretación sobre cómo y qué tanto han sido satisfechos los criterios de nuestro interés.

6. **Toma de decisiones**. La toma de decisiones realizada a partir del juicio construido constituye sin duda el porqué y para qué de la evaluación. Las decisiones que se tomen en la evaluación pueden ser de dos tipos: de carácter **estrictamente pedagógico** (para lograr ajustes y mejoras necesarias de la situación de aprendizaje y/o de enseñanza) y de **carácter social** (las cuales tienen que ver con asuntos como la acreditación, la promoción, etc.).

Este último punto nos lleva definir las funciones que tiene la evaluación y que no debemos de dejar a un lado en dicho proceso.

La **función pedagógica** tiene que ver directamente con la comprensión, regulación y mejora de la situación de enseñanza y aprendizaje. Se evalúa para obtener información que permita, en un momento determinado, saber qué pasó con las estrategias de enseñanza y cómo es que están ocurriendo los aprendizajes de los alumnos, para que en ambos casos sea posible realizar las mejoras y ajustes necesarios.

La **función pedagógica** se integra al proceso de enseñanza como una genuina **evaluación continua**, dirigida a tomar decisiones de índole pedagógica y que verdaderamente justifica o le da sentido a la evaluación.

La **función social** de la evaluación se refiere a los usos que se dan de ésta más allá de la situación de enseñanza y aprendizaje, y que tienen que ver con cuestiones tales como la selección, la promoción, la acreditación, la certificación y la información a otros.

Se deben tener presentes dos **cuestiones para evaluar el proceso de construcción**:

1. Es necesario tratar de valorar todo el **proceso en su dinamismo**; las evaluaciones que sólo toman en cuenta un momento determinado resultarán más limitadas que aquellas otras que tratan de apreciar distintas fases del proceso.

2. El proceso de construcción no puede explicarse en su totalidad partiendo exclusivamente de las acciones cognitivas y conductuales de los alumnos, de las acciones docentes en su más amplio sentido (actividades de planeación, de enseñanza y hasta las evaluativas) ya que los factores contextuales del aula también desempeñan un papel importante y quizá decisivo.

El interés del profesor al **evaluar los aprendizajes** debe residir en:

- El grado en que los alumnos han construido, gracias a la ayuda pedagógica recibida y al uso de sus propios recursos cognitivos, interpretaciones significativas y valiosas de los contenidos revisados.

- El grado en que los alumnos han sido capaces de atribuir un valor funcional a dichas interpretaciones.

Apéndice A: Recursión

La presentación de la **Unidad de Recursión**, que forma parte de programa oficial de la asignatura de **Programación III**, "**Estructuras de Datos**"; es una porción de los resultados del diplomado **FORMACIÓN Y ACTUALIZACIÓN PARA UN NUEVO MODELO EDUCATIVO**.

Estos, serán los apuntes, sobre los cuales se abordará la clase, sin que necesariamente se les entreguen a lo alumnos, pues ellos, al finalizar la unidad, tendrán la capacidad de escribir sus propias notas sobre la recursión.

El **Objetivo de Asignatura** es el siguiente: El alumno analizará las estructuras de datos con las que se logra un uso óptimo de los recursos de la computadora, y aplicará las técnicas de programación para escribir programas útiles y portables, que implementen y manejen estas estructuras.

El **Objetivo Particular** de la **Unidad de Recursión** es: El alumno aplicará los conceptos de funciones recursivas para resolver problemas de programación, y observará que no todos los problemas se pueden resolver en un tiempo razonable.

En el **Apéndice B**, se muestra un cuestionario que tendrán que resolver los estudiantes, para hacer suyo el aprendizaje y puedan autoevaluar su propio aprendizaje.

El **contenido** de esta **Unidad Didáctica** es el siguiente:

CONTENIDO

a.1 Recursión

La **recursión**, es la propiedad de una función de llamarse a sí misma **[Burden 02, Chapra 96, Heileman 98, Nakamura 92 y Nieves 99]**. En matemáticas, la recursión se presenta cuando una función está definida en términos de sí misma. La idea de hacer una función recursiva, es que la función pueda ser expresada en unas pocas líneas. La **recursión** es un método general de resolver los problemas reduciéndolos a problemas más simples de un tipo similar (estrategia **Divide y Vencerás**). Por ejemplo tenemos la función:

$$f(x)=\begin{cases} 0 & x=0 \\ 2*\ f(x-1)+x^2 & x>0 \end{cases}$$

La función anterior puede ser evaluada para algunos valores de **x**, por ejemplo: `f(0)=0, f(1)=1, f(2)=6, f(3)=21,` etc. Este tipo de funciones manejan, lo que se conoce como **caso base**, es decir, un valor para el cual el valor de la función es conocido, sin necesidad de recursión.

Cuando una función en **lenguaje C** llama a otra función, tiene que comunicar el valor actual ó la posición de todas las variables y parámetros a la nueva función. Para realizar esto, hay que colocar estos valores en una **pila**.

Si otra función necesita resultados de una tercera, es necesario guardar otra vez las variables en la **pila**. Sin embargo, este proceso no requiere que una función conozca lo que se hace en la otra, es decir, este sistema es el que hace posible la recursión. Al escribir una función recursiva, se deben de tomar en cuenta las **cuatro reglas** siguientes:

- **Caso base**.- siempre se debe de tener uno o más casos base, los cuales se puedan resolver sin recursión.
- **Progreso**.- para los casos que se resuelvan recursivamente, la llamada recursiva siempre debe de tener un **caso base**.
- **Regla de diseño**.- se supone que todas las llamadas recursivas funcionan.
- **Regla de interés compuesto**.- el trabajo nunca se debe de duplicar resolviendo el mismo problema en llamadas recursivas separadas.

Los **problemas** con la recursión residen en los costos de su funcionamiento. Aunque estos costos casi siempre son justificables por que los programas, no solo por que son más pequeños, sino también por que son más claros, la *recursión nunca debe de usarse para sustituir un solo ciclo*. Muchas estructuras de datos usan recursión, como ejemplo tenemos: ordenación por fusión, ordenación rápida ó QuickSort, árboles binarios, fractales, etc.

a.2 Ejemplos de funciones recursivas

Suma de enteros no negativos.- este ejemplo solo es ilustrativo, pues todos los lenguajes de programación tienen esta operación, implementada de forma más eficiente.

$$suma(a,b)=\begin{cases} a & b=0 \\ suma(a,b-1)+1 & b>0 \end{cases}$$

```
int suma(int a, int b){
if(b == 0) return(a);
else return(suma(a,b - 1) + 1);
}
```

Para **a = 2** y **b = 3**:

Recursión	a,b	Llamada	Retorno
1	2,3	`suma(2,2) + 1`	**2 + 1 +1**+1
2	2,2	`suma(2,1) + 1`	**2 + 1** + 1
3	2,1	`suma(2,0) + 1`	**2** + 1

Se observa que la cantidad de llamadas recursivas es **b**.

Multiplicación.- este ejemplo solo es ilustrativo, pues todos los lenguajes de programación tienen esta operación, implementada de forma más eficiente.

$$mult(a,b) = \begin{cases} 0 & b = 0 \\ mult(a,b-1) + a & b > 0 \end{cases}$$

```
/* Recursiva */
int mult(int a,int b){
if(b == 0) return(0);
else return(mult(a,b - 1) + a);
}
```

```
/* Iterativa */
int mult(int a,int b){
int c = 0,i;
for(i = b; i > 0;i--) c = c + a;
return(c);
}
```

Para **a = 2** y **b = 3**:

Recursión	a,b	Llamada	Retorno
1	2,3	mult(2,2) + 2	**0 + 2 + 2** + 2
2	2,2	mult(2,1) + 2	**0 + 2** + 2
3	2,1	mult(2,0) + 2	**0** + 2

Iteración	c
1	2
2	4
3	6

Se observa que la cantidad de **llamadas recursivas**es **b**. El número de **iteraciones** es **b**. Sin embargo la multiplicación es factible en un solo paso.

Factorial.- la función factorial es un ejemplo claro del uso de la recursión, sin embargo se considera que es más eficiente el empleo de técnicas iterativas. Para calcular el factorial de un número **n** es necesario realizar el producto sucesivo de **1*2*3*…*n**, por definición para **0! = 1**.

$$fact(n) = \begin{cases} 1 & n = 0 \\ fact(n-1) * n & n > 0 \end{cases}$$

```
/* Factorial Recursiva */
double factorial(double n){
if(n == 0) return(1);
else return(n * factorial(n - 1));
}
```

```
/* Factorial Iterativa */
double factorial(double n){
int i;
double fact = 1;
for(i = 1;i <= n;i++)
      fact = fact * i;
return(fact);
}
```

Para **n = 4**:

Recursión	n	Llamada	Retorno
1	4	4* factorial(3)	**1*1*2*3***4
2	3	3* factorial(2)	**1*1*2***3
3	2	2* factorial(1)	**1*1***2
4	1	1* factorial(0)	**1***1

Iteración	factorial
1	1
2	1*2
3	1*2*3
4	1*2*3*4

Se observa que la cantidad de **llamadas recursivas** es **n**. El número de **iteraciones** es **n**.

Números de Fibonacci.- este un problema que puede ser resuelto fácilmente usando la recursión. La secuencia de números **Fibonacci** es de la forma: **0, 1, 1, 2, 3, 5, 8, 13, 21,…** La fórmula general es:

$$fib(n) = \begin{cases} 0 & n \leq 1 \\ 1 & n = 2 \\ fib(n-1) + fib(n-2) & n > 2 \end{cases}$$

En este caso se tienen dos casos base, uno para `fib(1) = 0` y otra para `fib(2) = 1`, de otra forma, la llamada recursiva se encontraría indefinida a partir de `n = 3`.

```
/* Fibonacci Recursiva */
int fib(int n){
if(n <= 1) return(0);
if(n == 2) return(1);
return(fib(n-1) + fib(n-2));
}
```

```
/* Fibonacci Iterativa */
int Fib(int n){
int i,a = 0,b = 1,c;
if(n <= 1) return(a);
if(n == 2) return(b);
for(i = 2;i < n;i++) {
    c = a + b; a = b; b = c;
    }
return(c);
}
```

Para **n = 5**:

Recursión	Llamada
1	**fib(4) + fib(3)**
2	**fib(3) + fib(2) + fib(2) + fib(1)**
3	**fib(2) + fib(1)** + 1 + 1 + 0

Iteración	c
1	1
2	2
3	3

El numero de funciones recursivas crece **exponencialmente**, por lo que el uso de la recursión en este caso es inadecuado. El número de llamadas recursivas para algunos valores de **n**, se muestra en la **Tabla a.1** y en la **Figura a.1**.

n	Fib(n)	Llamadas
5	3	8
6	5	14
7	8	24
8	13	40
9	21	66
10	34	108
11	55	176
12	89	286
13	144	464
14	233	752
15	377	1,218

n	Fib(n)	Llamadas
16	610	1,972
17	987	3,192
18	1,597	5,166
19	2,584	8,360
20	4,181	13,528
21	6,765	21,890
22	10,946	35,420
23	17,711	57,312
24	28,657	92,734
25	46,368	150,048

Tabla a.1: Secuencia **Fibonacci** y número de llamadas recursivas.

El número de llamadas recursivas para `Fibonacci(n)` es:

Llamadas = (2 / Ö5) * (Æ n + (1 / Æ) n) – 2

Donde: **Æ = (1 + Ö5) / 2**

El número de llamadas iterativas es **n - 2**.

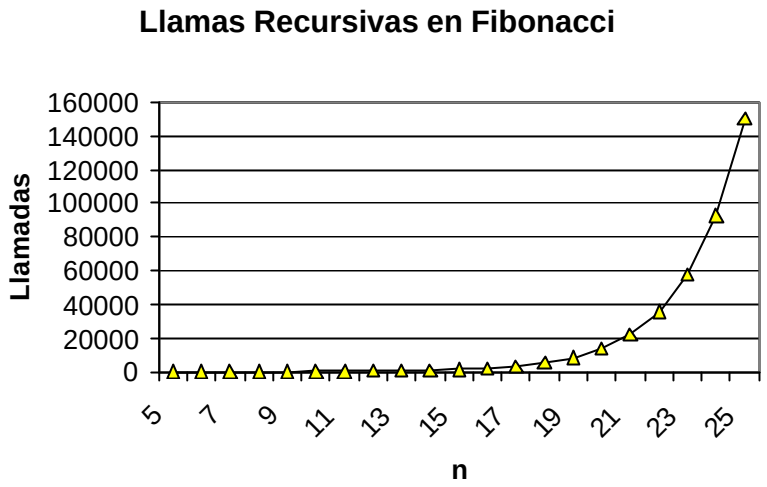


Figura a.1: Número de llamadas para la secuencia Fibonacci.

Existe una función más eficiente, incluso que la iteración (Formula analítica). Está es una fórmula que permite obtener la secuencia de **Fibonacci** para n ≥ 5:

$$fib(n) = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^{n-1} - \left(\frac{1-\sqrt{5}}{2}\right)^{n-1}\right)$$

a.3 Torres de Hanoi

Hasta ahora, los ejemplos vistos, es posible resolverlos de manera más eficiente de forma iterativa y hemos usado la recursión para ejemplificar su uso. Sin embargo; hay problemas, los cuales sería muy difícil resolverlos por métodos iterativos, el ejemplo más famoso de estos problemas, es el de las **Torres de Hanoi**, el cual es descrito a continuación:

En el gran templo de Benares, bajo la cúpula que marca el centro del mundo, descansa una placa de bronce en la que se encuentran fijadas tres agujas de diamante, altas y tan delgadas como el cuerpo de una abeja. En una de estas agujas Dios colocó, en la creación sesenta y cuatro discos de oro puro, el mayor descansando sobre la placa de bronce y los demás en tamaño decreciente hacia arriba. Esta es la torre de Brama. Día y noche sin parar, los sacerdotes transfieren los discos de una aguja de diamante a otra de acuerdo a las leyes fijas e inmutables de Brama, que indican que el sacerdote de servicio no debe mover más de un disco cada vez y que puede colocarlo en una aguja en la que no haya un disco de tamaño menor. Cuando se hayan transferido los sesenta y cuatro discos de la aguja en la que Dios los colocó en la Creación a otra de las agujas, torre, templo y Brahamanes se desmoronarán y se convertirán en polvo y, con un trueno, el mundo se desvanecerá.

Es decir, para **n** como el **número de discos** y tres columnas **A, B** y **C**, el programa debe de seguir estas reglas:

1. Si **n = 1**, mover el disco único de **A** a **B** y parar.
2. Mover el disco superior de **A** a **C**, **n-1** veces usando **B** como auxiliar.
3. Mover el disco restante de **A** a **C**.
4. Mover los discos **n-1** de **C** a **B** usando **A** como auxiliar.

El programa que realiza estas operaciones es el siguiente:

```
/* Torres de Hanoi                                                    */
void hanoi(int n,char p1,char p2,char p3){
if(n == 1) {
      printf("\tMover disco %d del poste %c al poste %c\n",1,p1,p2);
      return;
      }
hanoi(n - 1,p1,p3,p2);
printf("\tMover disco %d del poste %c al poste %c\n",n,p1,p2);
hanoi(n - 1,p3,p2,p1);
}

/* Programa principal                                                 */
void main(){
int n;
printf("\n\t\tProblema de las Torres de Hanoi\n");
n = 3;
hanoi(n,'A','B','C');
}
```

Programa a.1: Problema de las Torres de Hanoi.

En la **Figura a.3** se muestran los movimientos que se tienen que realizar para mover los tres discos, de acuerdo a las reglas anteriores.

```
Problema de las Torres de Hanoi para 3 discos
Mover disco 1 del poste A al poste B
Mover disco 2 del poste A al poste C
Mover disco 1 del poste B al poste C
Mover disco 3 del poste A al poste B
Mover disco 1 del poste C al poste A
Mover disco 2 del poste C al poste B
Mover disco 1 del poste A al poste B
```

Figura a.2: Resultado del programa para las Torres de Hanoi para 3 discos.

Como se observa, el número de movimientos para **n** discos, es $2^n - 1$. La **Tabla a.2**, muestra el tiempo de necesario para mover los discos, para algunos valores de **n.** (**mps** = movimientos por segundo).

n	1,000 mps	1,000,000 mps
8	255 ms	255 µs
16	65.5 s	66 ms
32	49.71 días	71.58 min
64	584,942 milenios	585 milenios

Tabla a.2: Tiempo para resolver el problema de las **Torres de Hanoi**.

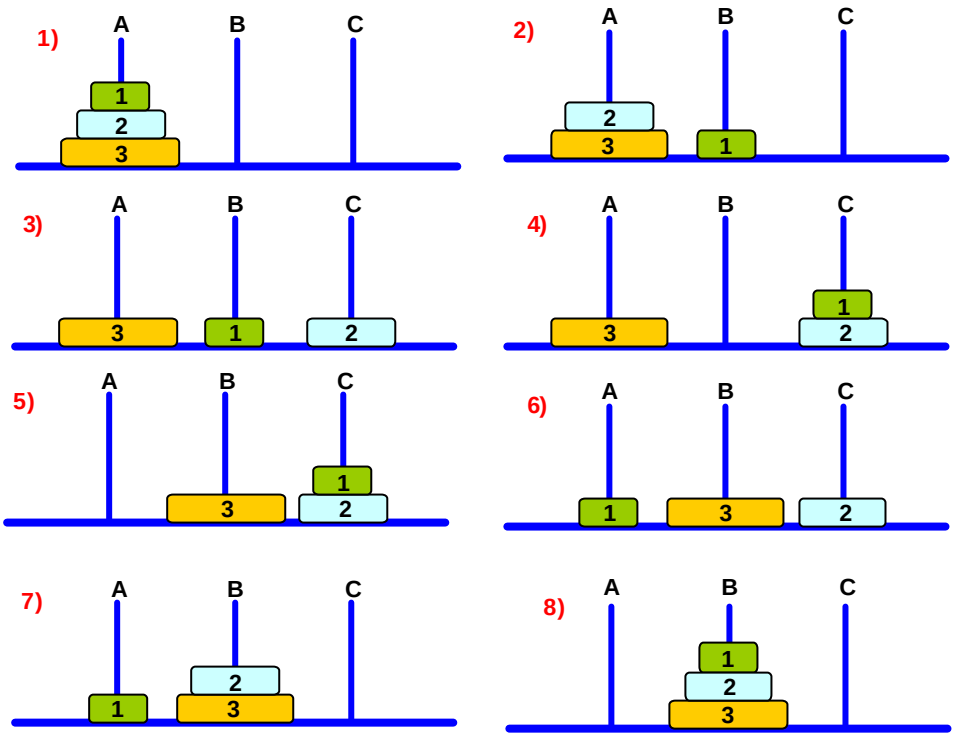


Figura a.2: Torres de Hanoi para 3 discos.

a.4 Función de Ackerman

La función de **Ackerman**, es una función recursiva, la cual está definida para valores enteros no negativos, **n** y **m**, de la forma:

A(0,n) = n + 1	**para m = 0 y n ³ 0**
A(m,0) = A(m - 1,1)	**para m ³ 0 y n = 0**
A(m,n) = A(m - 1, A(m,n - 1))	**para m > 0 y n ³ 0**

El código para resolver la función de **Ackerman** es el siguiente:

```
float ack(int m,int n){
if(m == 0) return(n + 1);
else  if (n == 0) return(ack(m - 1,1));
else return(ack(m - 1, ack(m,n - 1)));
}
```

En la **Tabla a.3** se muestran algunos valores y el número de llamadas recursivas para algunas evaluaciones de la función de **Ackerman**.

Función	Valor	Llamadas recursivas
A(1,1)	3	4
A(1,2)	4	6
A(1,3)	5	8
A(1,4)	6	10
A(2,1)	5	14
A(3,1)	13	106
A(4,1)	*	*
A(2,2)	7	27
A(2,3)	9	44
A(3,3)	61	2,432
A(3,4)	125	10,307

Tabla a.3: Valores y número de llamadas para la función de **Ackerman**.

Esta función también crece de forma extremadamente rápida, por lo que para algunos valores la cantidad de recursos disponibles en una computadora, no es suficiente para su cálculo.

a.5 Búsqueda

La tarea de búsqueda es una de las más frecuentes en análisis de datos. La suposición básica de un algoritmo de búsqueda, es tener una colección de datos, entre los cuales se buscará un elemento determinado. Por ejemplo, para un conjunto de datos en un arreglo, la tarea de búsqueda consiste en encontrar el índice del arreglo, en donde se encuentra el dato buscado, y regresar no encontrado, si el dato no esta en el arreglo.

a.5.1 Búsqueda secuencial ó lineal

Cuando no se brinda mayor información acerca de los datos buscados, entonces se hace imprescindible usar la búsqueda lineal, es decir, realizar una búsqueda verificando cada elemento del conjunto de elementos en donde probablemente se encuentre el elemento buscado (estrategia de **Fuerza Bruta**). Para terminar esta búsqueda, la cual se realiza elemento a elemento, se podrían dar cualquiera de las siguientes condiciones:

- Se halló el elemento buscado.
- Se recorrió todo el arreglo y no se encontró una concordancia.

El procedimiento para realizar una búsqueda lineal, en un arreglo **a[ ]**, con un numero **n** de elementos y si el dato que se busca es **x**, es el siguiente:

```
i = -1;
do i++;
while(a[i] != x && i <= n);
```

Si **i = n + 1**, entonces el elemento buscado no se encuentra en el arreglo. Este tipo de búsqueda es recomendado, en aquellos casos en donde no exista ningún tipo de orden en los datos, se presenta por ejemplo, cuando los datos se almacenan de forma aleatoria. Este tipo de búsqueda requiere de hasta **n** comparaciones.

a.5.2 Búsqueda Binaria

En la **búsqueda binaria**, los datos están ordenados, con lo que se aumenta la eficiencia de la búsqueda. Por ejemplo tenemos un directorio telefónico ó un diccionario, en donde los datos se encuentran ordenados de forma alfabética. Si no se procediera a darles un orden alfabético, el diccionario o el directorio telefónico serían inoperables.

La idea es entonces que dado un elemento **x** elegido para su búsqueda, compararlo con un elemento del arreglo elegido al azar, este puede ser el elemento que se encuentra a la mitad del arreglo, si ambos coinciden se termina la búsqueda, si es menor que **x**, se busca hacia la parte superior y si es mayor que **x** se busca hacia la parte inferior, siempre tratando de empezar cada nueva búsqueda por la parte central del subarreglo restante.

Por ejemplo tenemos el siguiente arreglo de datos:

Datos	2	3	5	7	15	18	23	31	36	48	56
Índice	0	1	2	3	4	5	6	7	8	9	10

Para encontrar el elemento x = 48 se sigue el siguiente procedimiento:

1) Seleccionamos la mitad del arreglo: **mitad = (0 + 10) / 2 = 5**, y comparamos: **x** con el dato del índice **5**.

Datos	2	3	5	7	15	**18**	23	31	36	48	56
Índice	0	1	2	3	4	**5**	6	7	8	9	10

Como **48** es mayor a **18** seguimos con el proceso, tomando la parte superior del arreglo.

2) Seleccionamos la mitad del arreglo: **mitad = (6 + 10) = 8**, y comparamos: **x** con el dato del índice **8**.

Datos	2	3	5	7	15	18	23	31	**36**	48	56
Índice	0	1	2	3	4	5	6	7	**8**	9	10

Como **48** es mayor a **36** seguimos con el proceso, tomando la parte superior del arreglo.

3) Seleccionamos la mitad del arreglo: **mitad = (9 + 10) = 9**, y comparamos: **x** con el dato del índice **9**.

Datos	2	3	5	7	15	18	23	31	36	**48**	56
Índice	0	1	2	3	4	5	6	7	8	**9**	10

Como **48** es igual a **48**, por lo que terminamos el proceso, pues encontramos el dato buscado en **índice = 9**.

En solo tres comparaciones encontramos el elemento buscado, por lo que se observa un enorme incremento en la eficiencia, respecto a la búsqueda lineal, la cual hubiese requerido de nueve comparaciones. El número promedio de comparaciones para este tipo de búsqueda es $\log_2 n$. El algoritmo que realiza este tipo de búsqueda se muestra en el **Programa a.2**.

```
#include "conio.h"
int a[] = {0,2,3,4,5,6,7,8,9,10};
int x;

/* Búsqueda Binaria                                                    */
int busqueda_binaria(int min,int max){
int med;
if(min > max) return(-1);
med =(min + max) / 2;
if(x == a[med]) return(med);
if(x > a[med]) return(busqueda_binaria(med + 1,max));
else return(busqueda_binaria(min,med - 1));
}
```

Programa a.2: Búsqueda Binaria (Parte **1**/2).

```
/* Programa principal */
void main(){
int y;
x = 10;
y = busqueda_binaria(0,10);
if(y >= 0)printf("\n\ta[%d] = %d",y,a[y]);
else printf("\n\n\t%d No esta en el arreglo.",x);
}
```

Programa a.2: Búsqueda Binaria (Parte **2**/2).

a.6 Definición recursiva de expresiones algebraicas

Una cadena algebraica es una cadena recursiva, en donde cada elemento se define de forma recursiva dadas las siguientes definiciones:

- **Expresión**.- es un término seguido por un signo más seguido por un término, o un término solo.
- **Término**.- es un factor seguido por un asterisco seguido por un factor o un factor solo.
- **Factor**.- es una letra o una expresión encerrada entre paréntesis.

Un programa recursivo, puede determinar si una cadena es válida, de acuerdo a las definiciones anteriores. El **Programa a.3** muestra un programa que acepta expresiones del tipo **a*(b+c)** y determina si es válida o no.

```
#include "stdio.h"
#include "ctype.h"
#define TRUE      1
#define FALSE     0
#define MAX 100
char lee_simb(char cad[],int largo,int *p);
char factor(char cad[],int largo,int *p);
char term(char cad[],int largo,int *p);
char expr(char cad[],int largo,int *p);

/* Regresa un carácter o un espacio                                  */
/* cad = entrada de la cadena de caracteres                          */
/* largo = longitud de cad                                           */
/* *p = es la ultima posición usada de cad                           */
char lee_simb(char cad[],int largo,int *p){
char c;
if(*p < largo) c = cad[*p];
else c = ' ';
(*p)++;
return(c);
}

/* Regresa verdadero si es un factor                                 */
/* cad = entrada de la cadena de caracteres                          */
/* largo = longitud de cad                                           */
/* *p = es la ultima posición usada de cad                           */
char factor(char cad[],int largo,int *p){
int c;
if((c = lee_simb(cad,largo,p)) != '(') return(isalpha(c));
return(expr(cad,largo,p) && lee_simb(cad,largo,p) == ')');
}
```

Programa a.3: Verificador de cadenas algebraicas (Parte **1**/2).

```
/* Regresa verdadero si es un Término                                    */
/* cad = entrada de la cadena de caracteres                              */
/* largo = longitud de cad                                               */
/* *p = es la ultima posición usada de cad                               */
char term(char cad[],int largo,int *p){
if(factor(cad,largo,p) == FALSE) return(FALSE);
if(lee_simb(cad,largo,p) != '*') {
      (*p)--;
      return(TRUE);
      }
return(factor(cad,largo,p));
}

/* Regresa verdadero si es una expresión                                 */
/* cad = entrada de la cadena de caracteres                              */
/* largo = longitud de cad                                               */
/* *p = es la ultima posición usada de cad                               */
char expr(char cad[],int largo,int *p){
if(term(cad,largo,p) == FALSE) return(FALSE);
if(lee_simb(cad,largo,p) != '+') {
      (*p)--;
      return(TRUE);
      }
return(term(cad,largo,p));
}

/* Programa principal                                                    */
void main(){
char cad[MAX];
int largo,pos;
printf("Verifica que una expresión algebraica sea válida.");
printf("Al terminar la expresión presione ENTER.");
printf("\n\n\t> ");
largo = 0;
while((cad[largo++] = getchar()) != '\n');
cad[--largo]='\0';
pos = 0;
if(expr(cad,largo,&pos) == TRUE && pos >= largo) printf("Cadena Válida");
else printf("Cadena no válida");
}
```

Programa a.3: Verificador de cadenas algebraicas (Parte **2**/2).

Al ejecutar el **Programa a.3** tenemos la salida de la **Figura a.3**.

```
Verifica que una expresión algebraica sea válida.
Al terminar la expresión presione ENTER.
> (a*b+c*d)+(e*(f)+g)
Cadena Válida
> (a*b+c*d)+(e*(f)+g
Cadena no válida
```

Figura a.3: Verificador de cadenas algebraicas.

a.7 Conversión prefija a postfija

Se trata de convertir una cadena prefija a su forma postfija (**Programa a.4**). El algoritmo que realiza dicha conversión es el siguiente:

- Si la cadena prefija es de una sola variable, esta es su propia equivalencia postfija.
- Se busca al primer operador de la cadena prefija.
- Se encuentra el primer operando de la cadena y se convierte a postfija.
- Se encuentra el segundo operando de la cadena y se convierte a postfija.
- Se juntan las cadenas postfijas encontradas.

```
#include "stdio.h"
#include "ctype.h"
#include "string.h"
#define MAX 100
void pp(char prefija[],char postfija[]);

/* Recorta cad entre n y m y regresa op1                                */
char substr(char cad[],int n,int m,char op1[]){
int i,j = 0;
for(i = n;i <= m;i++) op1[j++] = cad[i];
op1[j] = '\0';
return(*op1);
}

/* Realiza la conversión prefija a postfija                             */
void pp(char prefija[],char postfija[]){
char f[2],t1[MAX],t2[MAX];
f[0] = prefija[0];
f[1] = '\0';
substr(prefija,1,strlen(prefija)-1,prefija);
if(f[0] == '+' || f[0] == '-' ||  f[0] == '*' ||  f[0] == '/' ){
      pp(prefija,t1);
      pp(prefija,t2);
      strcat(t1,t2);
      strcat(t1,f);
      substr(t1,0,strlen(t1),postfija);
      return;
      }
postfija[0] = f[0];
postfija[1] = '\0';
}

/* Programa principal                                                   */
void main(){
char prefija[MAX],postfija[MAX];
int pos = 0;
printf("Convierte una Cadena Prefija a Postfija.");
printf("\n\n\t> ");
while((prefija[pos++] = getchar()) != '\n');
prefija[--pos] = '\0';
printf("Cadena prefija: %s",prefija);
pp(prefija,postfija);
printf("Cadena Postfija: %s",postfija);
}
```

Programa a.4: Conversión prefija a postfija.

Al ejecutar el **Programa a.4** tenemos la salida de la **Figura a.4**.

```
Convierte una Cadena Prefija a Postfija.
Al terminar la expresión presione ENTER.
> +a*bc
Cadena Prefija:  +a*bc
Cadena Postfija: abc*+
```

Figura a.4: Conversión prefija a postfija.

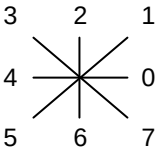
a.8 Fractales

Una aplicación de los procesos recursivos, se encuentra en la elaboración de graficas fractales. Regularmente se forman por la superposición repetida de un patrón grafico simple.

Curvas de Hilbert

El procedimiento de elaboración, consiste en dibujar líneas de una longitud definida, según una serie de reglas. Estas curvas reciben su nombre en honor al matemático que las invento en 1891. Las reglas son las que se muestran a continuación, en donde, cada flecha indica la dirección de la línea y cada letra una función:

Inicio A					
	A:	D ←	A ↓	A →	B
	B:	C ↑	B →	B ↓	A
	C:	B →	C ↑	C ←	D
	D:	A ↓	D ←	D ↑	C



Por ejemplo para la **curva de Hilbert** de orden 1 y 2, aplicamos las definiciones y se generan las curvas que se muestran en la **Figura a.5**.

Figura a.5: **Curvas de Hilbert** de orden 1 y 2.

El **Programa a.5** muestra, como se codifica en **C**, este procedimiento.

```
#include "graphics.h"
int u,x0,y0;
void a(int);
void b(int);
void c(int);
void d(int);

/* Imprime línea con dirección y tamaño                              */
void linea(int dir,int tam){
int x1,y1;
if(dir == 0){
      x1 = x0 + tam;
      y1 = y0;
      }
```

Programa a.5: Curvas de Hilbert (Parte **1**/3).

```
if(dir == 2){
      x1 = x0;
      y1 = y0 - tam;
      }
if(dir == 4){
      x1 = x0 - tam;
      y1 = y0;
      }
if(dir == 6){
      x1 = x0;
      y1 = y0 + tam;
      }
line(x0,y0,x1,y1);
x0 = x1;
y0 = y1;
}

void a(int i){
if(i > 0){
      d(i - 1);
      linea(4,u);
      a(i - 1);
      linea(6,u);
      a(i - 1);
      linea(0,u);
      b(i - 1);
      }
}

void b(int i){
if(i > 0){
      c(i - 1);
      linea(2,u);
      b(i - 1);
      linea(0,u);
      b(i - 1);
      linea(6,u);
      a(i - 1);
      }
}

void c(int i){
if(i > 0){
      b(i - 1);
      linea(0,u);
      c(i - 1);
      linea(2,u);
      c(i - 1);
      linea(4,u);
      d(i - 1);
      }
}
```

Programa a.5: Curvas de Hilbert (Parte **2**/3).

```
void d(int i){
if(i > 0){
      a(i - 1);
      linea(6,u);
      d(i - 1);
      linea(4,u);
      d(i - 1);
      linea(2,u);
      c(i - 1);
      }
}

void main(){
int graphdriver = DETECT,graphmode;
initgraph(&graphdriver,&graphmode,"");
cleardevice();
u = 30;
x0 = 400;
y0 = 80;
a(5);
getch();
closegraph();
}
```

Programa a.5: Curvas de Hilbert (Parte **3**/3).

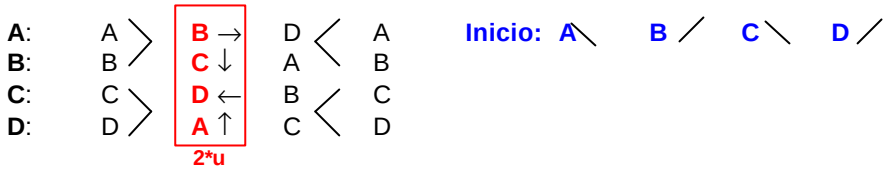
En la **Figura a.6** se observa la **curva de Hilbert** de orden **5**.

Figura a.6: Curva de Hilbert de orden 5.

El número de llamadas recursivas realizadas, coincide con el número de líneas dibujadas y es: $\mathbf{4^n - 1}$.

Curvas Sierpinski

Es otro procedimiento de superposición de curvas, la diferencia es que estas curvas son cerradas, el patrón es el siguiente:



En donde las líneas verticales y horizontales son del doble de tamaño. Por ejemplo, para las **Curvas Sierpinski** de orden 1 y 2 se muestran en la **Figura a.7**.

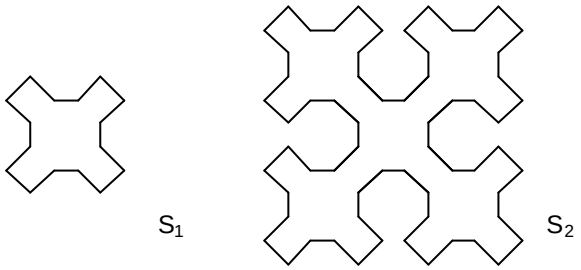


Figura a.7: Curvas Sierpinski de orden 1 y 2.

El **Programa a.6** muestra, como se codifica en C, este procedimiento.

```
#include "graphics.h"

int u,x0,y0;
void a(int);
void b(int);
void c(int);
void d(int);

void linea(int dir,int tam){
int x1,y1;
if(dir == 0){
      x1 = x0 + tam;
      y1 = y0;
      }
if(dir == 1){
      x1 = x0 + tam;
      y1 = y0 - tam;
      }
if(dir == 2){
      x1 = x0;
      y1 = y0 - tam;
      }
if(dir == 3){
      x1 = x0 - tam;
      y1 = y0 - tam;
      }
```

Programa a.6: Curvas de Sierpinski (Parte **1**/3).

```
if(dir == 4){
      x1 = x0 - tam;
      y1 = y0;
      }
if(dir == 5){
      x1 = x0 - tam;
      y1 = y0 + tam;
      }
if(dir == 6){
      x1 = x0;
      y1 = y0 + tam;
      }
if(dir == 7){
      x1 = x0 + tam;
      y1 = y0 + tam;
      }
line(x0,y0,x1,y1);
x0 = x1;
y0 = y1;
}

void a(int k){
if(k > 0){
      a(k - 1);
      linea(7,u);
      b(k - 1);
      linea(0,2 * u);
      d(k - 1);
      linea(1,u);
      a(k - 1);
      }
}

void b(int k){
if(k > 0){
      b(k - 1);
      linea(5,u);
      c(k - 1);
      linea(6,2 * u);
      a(k-  1);
      linea(7,u);
      b(k - 1);
      }
}

void c(int k){
if(k > 0){
      c(k - 1);
      linea(3,u);
      d(k - 1);
      linea(4,2 * u);
      b(k - 1);
      linea(5,u);
      c(k - 1);
      }
}
```

Programa a.6: Curvas de Sierpinski (Parte **2**/3).

```
void d(int k){
if(k > 0){
      d(k - 1);
      linea(1,u);
      a(k - 1);
      linea(2,2 * u);
      c(k - 1);
      linea(3,u);
      d(k - 1);
      }
}

void main(){
int graphdriver = DETECT,graphmode;
initgraph(&graphdriver,&graphmode,"");
cleardevice();
u = 15;
x0 = 50;
y0 = 70;
a(5);
linea(7,u);
b(5);
linea(5,u);
c(5);
linea(3,u);
d(5);
linea(1,u);
getch();
cleardevice();
closegraph();
}
```

Programa a.6: Curvas de Sierpinski (Parte **3**/3).

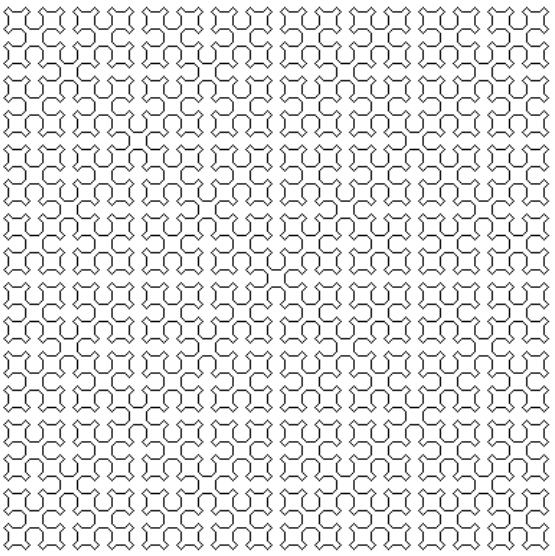


Figura a.8: Curva Sierpinski de orden 5.

El número de llamadas recursivas realizadas, coincide con el número de líneas dibujadas y es: 4^{n+1}.

a.9 El problema de las ocho Reinas

Este es un **algoritmo de tanteo** o **rastreo inverso**. El problema es que hay que colocar ocho reinas en un tablero de ajedrez en una forma en que ninguna reina se coma a otra. El **Programa a.7** muestra la como es posible obtener todas (**92**) las soluciones a este problema. Y en el **Programa a.8** se observa una segunda solución a este problema; ambas soluciones realizan el mismo número de llamadas recursivas.

```
#include "stdio.h"
#include "math.h"

int a[10],b[20],c[20],x[10];

/* Busca las soluciones de forma recursiva */
void reina(int i){
int i,j;
for(j = 1;j <= 8;j++){
      if(a[j] && b[i+j] && c[i – j + 8]){
            x[i] = j;
            a[j] = 0;
            b[i + j] = 0;
            c[i – j + 8] = 0;
            if(i < 8) reina(i + 1);
            else for(i = 1;i <= 8;i++) printf(" %d",x[i]);
            a[j] = 1;
            b[i+j] = 1;
            c[i – j + 8] = 1;
            }
      }
}

/* Programa Principal */
void main(){
int i;
for(i = 0;i <= 9;i++) a[i] = 1;
for(i = 0;i <= 18;i++) {
      b[i] = 1;
      c[i] = 1;
      }
reina(1);
}
```

Programa a.7: Problema de las **ocho Reinas**.

El programa entrega las posiciones en la fila correspondiente del tablero, las primeras seis soluciones se tienen en la **Figura a.9**. También se observa la distribución de las reinas en el tablero.

```
1:    1 5 8 6 3 7 2 4
2:    1 6 8 3 7 4 2 5
3:    1 7 4 6 8 2 5 3
4:    1 7 5 8 2 4 6 3
5:    2 4 6 8 3 1 7 5
6:    2 5 7 1 3 8 6 4
```

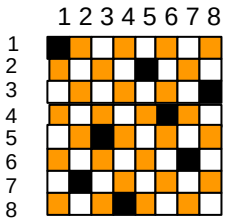


Figura a.9: Soluciones al problema de las **ocho Reinas**.

```
#include "stdio.h"
#include "conio.h"
#define     FALSE 0
#define     TRUE  1
#define     TT    8     /* Celdas del Tablero                          */

/* Variables Globales                                                  */
int tablero[TT][TT];

/* Válida una posición                                                 */
int valido(){
int i,j,k,l,m;
for(i = 0;i < TT;i++){
      for(j = 0;j < TT;j++){
            if(tablero[i][j]){
                  for(k = 0;k < TT;k++){
                        if(tablero[i][k] && k != j)
                              return(FALSE);
                        if(tablero[k][j] && k != i)
                              return(FALSE);
                              }
                        l = i + 1;
                        m = j + 1;
                        while(l < TT && m < TT){
                              if(tablero[l][m])
                                    return(FALSE);
                              l++;
                              m++;
                              }
                        l = i + 1;
                        m = j - 1;
                        while(l < TT && m > 0){
                        if(tablero[l][m])
                              return(FALSE);
                        l++;
                        m--;
                        }
                  }
            }
      }
return(TRUE);
}
```

Programa a.8: Problema de las **ocho Reinas** (Parte **1**/2).

```
/* Imprime el tablero de salida                                          */
void salida(){
int i,j;
for(i = 0;i < TT;i++){
      for(j = 0;j < TT;j++){
            if(tablero [i][j])printf("R ");
            else  cprintf("* ");
            }
      printf("\n");
      }
}

/* Busca la posición de las reinas en el tablero                         */
int reinas(int n){
int i;
for(i = 0;i < TT;i++) {
      tablero[n][i] = TRUE;
      if(n == TT - 1 && valido()) return TRUE;
      if(valido() && reinas(n + 1)) return TRUE;
      tablero[n][i] = FALSE;
      if(i == TT - 1) return(FALSE);
      }
return(TRUE);
}

/* Programa Principal                                                    */
void main(){
int i,j;
textbackground(1);
textcolor(14);
clrscr();
for(i = 0;i < TT;i++)
      for (j = 0;j < TT;j++)
            tablero[i][j] = FALSE;
if(reinas(0)) salida();
}
```

Programa a.8: Problema de las **ocho Reinas** (Parte **2**/2).

El programa entrega las posiciones en el tablero (**Figura a.10**).

```
R * * * * * * *
* * * * R * * *
* * * * * * * R
* * * * * R * *
* * R * * * * *
* * * * * * R *
* R * * * * * *
* * * R * * * *
```

Figura a.10: Solución al **problema de las ocho Reinas**.

Apéndice B: Preguntas de Repaso

En este apartado se muestra un cuestionario que tendrán que resolver los estudiantes, para hacer suyo el aprendizaje y puedan autoevaluar su propio aprendizaje.

I. Preguntas de auto evaluación

1. Defina la **recursión** desde el punto de vista:

   a) Matemático **b)** Informático

2. En **recursión**, que es el **caso base** .

3. Escriba las cuatro **reglas básicas de la recursión**.

4. Por que se considera que un **proceso iterativo** es más eficiente que un **proceso recursivo**.

5. Para cada uno de los siguientes ejemplos, determine el **número de llamadas recursivas** y el **número de ciclos iterativos**.

   a) suma(a,b) **b)** multiplicación(a,b) **c)** Factorial(n)

6. Defina y de un ejemplo del algoritmo de **búsqueda binaria**.

7. Defina una expresión algebraica como una **cadena recursiva**.

8. Defina el proceso de conversión de una **expresión prefija a postfija**.

9. Defina el patrón **fractal** para la curva de **Hilbert**.

10. Defina el patrón **fractal** para la curva **Sierpinski**.

II. Programas

1. Realice un programa que cuente el numero de llamadas recursivas para:

   a) suma(a,b) **b)** multiplicación(a,b) **c)** Factorial(n)

2. Realice un programa para evaluar la **función de Ackerman**, definida para dos enteros **m** y **n** de la forma:

   A(0,n) = n + 1 **para m = 0 y n ³ 0**
   A(m,0) = A(m - 1,1) **para m ³ 0 y n = 0**
   A(m,n) = A(m - 1, A(m,n - 1)) **para m > 0 y n ³ 0**

3. Realice un programa que cuente el numero de llamadas recursivas para:

   a) fibonacci(15) **b)** hanoi(16) **c)** ackerman(3,4)

4. Escriba un programa para verificar expresiones algebraicas simples.

5. Escriba un programa para obtener la solución al **problema de las ocho Reinas**.

Apéndice C: Sala iXtli

Como parte de la integración del uso de **tecnologías de la información y la comunicación** (**TIC**) en la práctica docente, se incluye una descripción del "**Observatorio de Visualización de la UNAM**", que es hoy en día una de las aulas que cuenta con la tecnología más avanzada, en cómputo, comunicaciones y electrónica, y en la cuál se han realizado algunas demostraciones realizadas para la asignatura descrita en el presente documento.

Contenido

Observatorio de Visualización de la UNAM

c.1 PRESENTACIÓN

iXtli, es el **Observatorio de Visualización de la UNAM**, es una sala de alta tecnología diseñada para visualizar y simular objetos complejos e imágenes en **tercera dimensión** (**3D**), mediante un sistema de **realidad virtual inmersiva**.

Este es un lugar de encuentro multidisciplinario, en el cual las nuevas tecnologías **computacionales** y de **electrónica** dan vida al trabajo docente y de investigación de los universitarios (apoderándose de nuestros sentidos y percepciones para crear una ilusión total de tridimensionalidad) posee las más avanzadas técnicas de **realidad virtual** para disposición de los académicos en la enseñanza y la investigación en todas las áreas del conocimiento humano.

La tecnología y el diseño de esta herramienta de trabajo permiten múltiples usos, lo que la hace única en México; además, es la sala con mayor capacidad de cómputo intensivo en operación, en una institución de educación superior en el país. En **iXtli** se puede ver, escuchar y tener una experiencia realmente innovadora a través de una pantalla curva, especialmente diseñada para realzar y mejorar las representaciones de los diferentes proyectos de investigación en el quehacer universitario y, sobretodo, para comprender mejor la realidad y los resultados de las investigaciones.

c.2 SALA

El elemento clave de la sala es su sistema de **Realidad Virtual Inmersiva**, cuyo propósito es lograr que los usuarios tengan la sensación de encontrarse dentro del mundo creado por computadora, esto se produce al generar, en tiempo real, **imágenes estereoscópicas** que el usuario percibe con profundidad y que, además, responden a sus órdenes o movimientos. Las imágenes calculadas en tiempo real se generan a partir de un modelo tridimensional almacenado previamente en el equipo gráfico de alto rendimiento que está a disposición de la sala.

Nuevas imágenes se generan siguiendo las órdenes y movimientos de los usuarios, los cuales, son capturados por medio de dispositivos de rastreo de movimiento, guantes o ratón tridimensionales. Asimismo, se cuenta con un sistema de sonido que proporciona sensaciones auditivas relacionadas con el ambiente creado.

c.2.1 CUPO

El espacio de **iXtli** está diseñado para albergar hasta **42** personas utilizando lentes estereoscópicos. Si el usuario lo requiere, es posible que un grupo de **6** u **8** investigadores puedan utilizar una mesa en el frente de la pantalla como espacio de trabajo para acceder a las diferentes funcionalidades de la sala.

c.2.2 PROYECTORES

Las imágenes son generadas por tres proyectores **Christie Digital Mirage 2000**, basados en tecnología de alta resolución **SXGA** (**1280x1024**). Cada proyector produce una tercera parte de la imagen y utiliza equipos de corrección de geometría y mezcla de imágenes, para producir una sola imagen de resolución **3520 x 1024**, que cubre la totalidad de la pantalla.

c.2.3 PANTALLA

Las imágenes son proyectadas en **mono** o en **estéreo** en una pantalla semicilíndrica de **140 grado**s, que mide **8.90 mts**. de **largo** por **2.55 mts.** de **ancho**. La forma **cilíndrica** permite cubrir gran parte del campo de visión del usuario lo que le ayuda a sentirse dentro del **ambiente virtual**.

c.2.4 ESTÉREO ACTIVO

La sensación de **profundidad** se genera al producir imágenes diferentes para el ojo derecho e izquierdo; éstas se proyectan en forma alternada con una alta velocidad. Los lentes de cristal líquido, **Crystal Eyes**, bloquean la visión de los ojos siguiendo la sincronía proporcionada por la computadora; así, cada ojo recibe solamente la imagen que le corresponde de manera tan rápida que engaña al cerebro y le hace creer que ve una sola imagen con **profundidad**.

c.2.5 DISPOSITIVOS

La sala cuenta con un sistema de rastreo de movimiento **InterSense IS-900** para detectar con precisión la posición y orientación de las partes del cuerpo, en relación con el modelo visualizado. Se tienen dos sensores, uno para el rastreo del movimiento de la cabeza y otro que rastrea la posición y movimientos de un **mouse tridimensional** o **wanda**.

Además, se cuenta con un **guante inalámbrico 5DT Data Glove**, con dos sensores por dedo, para medir tanto la **flexión** como la **abducción** entre los dedos.

c.2.6 AUDIO

La **sala iXtli** posee un sistema de sonido de alta tecnología:

Equipo **Dolby Surround 5.1** que a través de tres bocinas delanteras, dos bocinas traseras y un sub-woofer, envuelve al público con sonido tridimensional. Está equipada para reproducir formatos **CD**, **DVD-A**, **Wav** y **MP3**, entre otros.

c. 2.7 CÁMARAS

iXtli cuenta con tres cámaras robóticas en la parte superior de la pantalla que pueden integrarse a un sistema de videoconferencia o de utilizarse como dispositivos para interacción con algoritmos de visión por computadora.

c.2.8 SUPERVIEW

Este manejador de ventanas permite sobreponer diversas fuentes de video en la pantalla; por ejemplo, es posible tener como base una visualización de un modelo **3D** en estéreo, abarcando las tres pantallas y sobreponer una ventana con el video de una **PC**, mostrando un **Power Point** y, otra ventana con un **DVD** corriendo una animación pregrabada.

Al utilizar el **superview**, el usuario debe definir al menos una fuente de fondo, que puede ser la imagen generada por la **SGI** o la **PC**, en cualquiera de sus configuraciones. Posteriormente, se seleccionan las fuentes que se mostrarán como ventanas sobrepuestas, las cuales pueden moverse a través de toda la pantalla, modificar sus dimensiones, aparecerlas o desaparecerlas, según lo indique el usuario desde el sistema de control (**touchscreen**). Es posible tener hasta ocho ventanas abiertas al mismo tiempo. El tamaño máximo de una ventana es de **1280 x 1024** píxeles.

c.2.9 FUENTES DE IMAGEN

Las fuentes de imagen pueden ser:

SGI, **MAC**, **PC**, **DVD**, **VHS**, **VC** y **LAPTOP**.

c.2.10 GRABACIÓN Y REPRODUCCIÓN

DVD.- la sala ofrece la facilidad de grabar o desplegar, de manera simultánea, presentaciones directas en la pantalla. El equipo con que se cuenta es el siguiente:

PRV-9000 Pro DVD-Video Recorder

Recording format:	DVD Video, VR.
Recording media:	DVD-R, DVD-RW.
Playback format:	DVD Video, VR, Video CD, CD-Audio.
Playback media:	DVD-R, DVD-RW, CD-R, CD-RW.
Video input:	BNC, RCA.
Video output:	BNC. S-video input, output. Component video output.
Audio input:	BNC.
Audio output:	RCS.
DV input/output:	iLINK/IEEE 1394.

Tiempo de Grabado (DVD 4.7 GB)

1 Hora:	V1: DVD-Video Format
2 Horas:	V2: DVD-Video Format
2 Horas:	SP: Video Recording format
1-6 Horas:	MN: Video Recording format

VHS.- la sala ofrece la facilidad de grabar o desplegar, de manera simultánea, presentaciones directas en la pantalla. El equipo con que se cuenta es el siguiente:

World Wide CVR Samsung SV-7000W/XEAU
Worldwide NTSC, PAL, SECAM.
Worldwide tape conversion.
Worldwide TV Broadcasting Tuner

Format:	VHS standard TV System Input/Output: NTSC 3.58, NTSC 4.43, PAL, PAL-N, PAL-M, SECAM , MESECAM, SECAM-L
Tape Recording/Playback:	NTSC, PAL, PAL-M, SECAM, MESECAM, SECAM-L NTSC/PAL-M
Recording Speed:	SP/SLP NTSC/PAL-M/SECAM-L
Playback Speed:	SP/LP/SLP PAL/PAL-M/SECAM
Recording/Playback Speeds:	SP/LP

VIDEOCONFERENCIA.- Se cuenta con la tecnología para realizar videoconferencias a través de un **codec 6000** de **Tandberg Video Systems**. El **protocolo** utilizado en el sistema es **H.323** a través de la red **LAN** a **100 Mbps**. Se dispone de 5 micrófonos omnidireccionales los cuales cubren por completo la sala **IXTLI**.

En lo referente a las fuentes de video, se tiene la versatilidad de incluir a cualquiera de las señales de video que se manejan dentro de **IXTLI**. Gracias a la capacidad **multipunto** (**MCU**), el sistema puede establecer conferencias hasta con **4** participantes.

c.2.11 PANEL DE CONTROL

Desde la cabina de control o un **touchscreen móvil**, el usuario puede controlar y seleccionar los elementos de video, sonido, iluminación y control de **keyboards** de **iXtli**.

Personal del grupo de realidad virtual puede ajustar, programar y guardar configuraciones de uso cotidiano, según los requerimientos del usuario.

c.2.12 EQUIPO

Los modelos desplegados en la pantalla son generados en tiempo real desde una máquina **Onyx**, **PC** o **MAC**. Dependiendo de las necesidades del usuario, se puede hacer uso de aplicaciones instaladas en las computadoras de la sala o solicitar la instalación de nuevas aplicaciones, como son los desarrollos específicos del área en la que se esté trabajando.

c.2.13 CONECTIVIDAD

La sala cuenta con red inalámbrica que cubre el área del vestíbulo y el interior de la sala, esto permite realizar sesiones de trabajo con aquellos asistentes que utilizan **laptop** y desean conectarse a la red. Además, los equipos de cómputo **SGI Onyx 350** y los equipos auxiliares **PC**, **MAC**, y videoconferencia tienen conexión a **1 GB ethernet** al **backbone** de la **UNAM**. Esta conexión proporciona también acceso a **Internet 2** para los proyectos que requieren una red de tecnologías avanzadas.

c. 2.14 PREVISUALIZACIÓN

El área de previsualización es un espacio de trabajo que cuenta con tres monitores para los diferentes formatos de video desde la **SGI**, **PC** o **MAC**. Este espacio permite trabajar en tres pantallas con un formato similar al del interior de la sala **iXtli**, donde el investigador podrá visualizar sus aplicaciones, antes de mostrarlas dentro de la sala. Este espacio está ubicado a un costado del Departamento de Realidad Virtual.

c.3 HARDWARE

SGI Onyx 350

Tipo de procesador:	MIPS R16000 64-bit @ 700 Mhz, 4MB caché.
Número de procesadores:	12
Número de pipes gráficos:	3 con 2 RasterManager InfiniteReality4 cada uno
Memoria:	24 GB SDRAM
Almacenamiento:	2x73 GB, arreglo TP9100: 16x146 GB
Comunicaciones:	1 puerto 1000Base-T, 4 puertos seriales 115.2 kbauds, 4 puertos USB, 2 PS/2

Power MAC

Procesador:	POWERPC G5 Dual 2GHz
Caché L2:	512 K por procesador
Bus frontal:	1 GHz por procesador
Memoria Principal:	512 MB PC3200 DDR SDRAM
Tarjeta de video:	ATI RADEON 9600 Pro con 64 MB DDR SDRAM
Almacenamiento:	160 GB SATA
Unidad Óptica:	Superdrive (DVD-R/CD-RW)
Ranuras PCI-X (3):	(1) 133 MHz, 64-bit. (2) 100 MHz, 64-bit
I/O:	1 Puerto Firewire 800, 2 Puertos Firewire 400, 3 USB 2.0, 2 USB 1.1
Conectividad:	10/100/1000BASE-T Ethernet y módem 56 K V.92
Software de Sistema:	Mac OS X v10.2

DELL Precision 450

Procesador:	Intel Pentium XEON Dual
Bus Frontal:	533 Mhz
Cache L2:	512 K por procesador
Memoria:	2 GB DDR SDRAM 266 Mhz, NECC
Almacenamiento:	14 GB, SCSI 48x CD-RW/DVD
Tarjeta de video:	NVIDIA Quadro FX 1000, 128 MB
Comunicaciones:	Intel Gigabit Ethernet, 3 USB 2.0, 2 USB 1.1.

c.4 SOFTWARE

c.4.1 Visualización

Software	Plataforma	Disponibilidad	Mayor información
OpenDX	SGI IRIX, Linux, Windows 2000/XP	Freeware	www.opendx.org
VTK (Visualization Toolkit)	SGI IRIX, Linux, Windows 2000/XP	Freeware	www.vtk.org
VMD (Visual Molecular Dynamics)	SGI IRIX, Linux, Windows 2000/XP,Mac	Freeware	www.ks.uiuc.edu/Research/vmd/
Vis5d	SGI IRIX, Linux,Windows 2000/XP	Freeware	vis5d.sourceforge.net/
Ribbons	SGI IRIX, Linux,Windows 2000/XP,Mac	Freeware	sgce.cbse.uab.edu/ribbons/
Proteing Crystallographic Package	SGI IRIX	Freeware	origo.imsb.au.dk/~mok/o/#software
OpenSG	SGI IRIX, Linux, Windows 2000/XP	Freeware	www.opensg.org
OpenSceneGraph	SGI IRIX, Linux,Windows 2000/XP	Freeware	openscenegraph.sourceforge.net/
AVS	SGI IRIS	Comercial	www.avs.com/
Amira	SGI IRIS	Comercial	www.tgs.com

c.4.2 Modelado

Software	Plataforma	Disponibilidad	Mayor información
Alias Wavefront Maya	Alias Wavefront Maya	Comercial	www.aliaswavefront.com
Multigen Creator	Windows 2000/XP	Comercial	www.multigen.com/
Blender	SGI IRIX, Linux, Windows 2000/XP, Mac	Freeware	www.blender.org/
AC3D	SGI IRIX, Linux	Freeware	www.ac3d.org/

c.4.3 Convertidor de formatos

Software	Plataforma	Disponibilidad	Mayor información
Nugraf	Windows 2000/XP	Comercial	www.okino.com/
vtkActorToPF	SGI IRIX, Linux, Windows 2000/XP	Freeware	brighton.ncsa.uiuc.edu/~prajlich/vtkActorToPF/

c.4.4 Rendering

Software	Plataforma	Disponibilidad	Mayor información
IRIS Performer	SGI IRIX, Linux	Freeware	www.sgi.com/software/performer/
Blue Moon Rendering Tools	SGI IRIX, Linux, Windows 2000/XP	Freeware	www.bmrt.org
Alias Wavefront Maya	Windows 2000/XP	Comercial	www.aliaswavefront.com
Frame Cycler Digital Daily System	Windows 2000/XP	Comercial	stereoweb.iridas.com/
PovRay	Linux, Windows 2000/XP, Mac	Freeware	www.povray.org/
Radiance	SGI IRIX, Linux, Windows 2000/XP	Freeware	radsite.lbl.gov/radiance/HOME.html

c.4.5 Realidad Virtual

Software	Plataforma	Disponibilidad	Mayor información
VR Juggler	SGI IRIX, Linux, Windows 2000/XP	Freeware	www.vrjuggler.org
VR Nav	SGI, Linux, Windows 2000/XP	Freeware	www.ats.ucla.edu/at/vrNav/default.htm
Saranav	SGI IRIX, Linux	Freeware	www.ncsa.uiuc.edu/VR/cavernus/SARA/Saranav.html

c.4.6 Bibliotecas gráficas

Software	Plataforma	Disponibilidad	Mayor información
OpenInventor 4.0	SGI IRIS	Comercial	www.tgs.com/
OpenGL	SGI IRIX, Linux, Windows 2000/XP, Mac	Freeware	www.opengl.org
OpenInventor 2.1.5	Linux, SGI IRIS	Comercial	oss.sgi.com/projects/inventor/
GL4Java	Linux, Mac, Windows 2000/XP	Freeware	jausoft.com/gl4java/
Java3D	SGI IRIX, Linux, Windows 2000/XP	Freeware	java.sun.com/products/java-media/3D/
DirectX/Direct3D	Windows 2000/XP	Freeware	www.microsoft.com/DirectX
OpenGL Multipipe	SGI IRIX	Freeware	www.sgi.com/software/multipipe/
OpenGL Shader	SGI IRIX, Linux	Freeware	www.sgi.com/software/shader/

c.5 USOS

c.5.1 INMERSIÓN

En **iXtli** se utilizan técnicas de **realidad virtual inmersiva** (**RVI**) para crear una sensación de inmersión dentro de los datos y dentro de un mundo generado por computadora con el cual se puede interactuar. Este efecto se realiza al generar imágenes estereoscópicas, calculadas en tiempo real siguiendo las órdenes del usuario y una inmersión auditiva relacionada con el ambiente creado. Las aplicaciones que pueden hacer uso de la inmersión son aquéllas que obtienen beneficios de la libertad de interacción y de la sensación de presencia dentro del mundo tridimensional. Entre las disciplinas que han hecho uso de esta tecnología podemos mencionar; por ejemplo, la **Arquitectura**, donde la inmersión se ha utilizado para navegar los espacios que se diseñan, con el fin de sentir las dimensiones y adecuarlos antes de construirlos; en **Medicina**, los médicos utilizan estos sistemas para manipular órganos y observar funcionamiento, con el fin de enseñar o realizar técnicas de cirugía; por su parte **Físicos** o **Químicos** emplean estas técnicas cuando buscan valores o propiedades al interactuar con visualizaciones, resultados de simulaciones numéricas; mientras que los interesados en la búsqueda de nuevas formas de expresión crean diversos espacios con los medios digitales.

Así que la **Realidad Virtual Inmersiva** encuentra su aplicación en una importante gama del **conocimiento humano**, en las áreas **científicas**, **sociales**, de **humanidades** y las **artísticas**.

c.5.2 INTERACCIÓN

Las aplicaciones disponibles en **iXtli** están diseñadas para permitir una forma natural de interacción a través de los movimientos del usuario, esto se consigue con el uso de un sistema de **captura de movimiento ultrasónico**, un ratón tridimensional y un guante, responsables de capturar y enviar a la computadora la posición de diversos sensores, que la máquina interpreta y traduce en instrucciones. Con este tipo de herramientas **iXtli**, se convierte en el espacio adecuado para la investigación de **interfaces humano-computadora**, lo que da cabida a variados proyectos de investigación que se llevan a cabo en la **UNAM**.

c.5.3 DESPLIEGUE

Aun cuando el uso principal de **iXtli** es la inmersión, el amplio espacio de proyección permite llevar a cabo trabajos de comparación de resultados, al mostrar en cada una de las tres pantallas que posee diversas visualizaciones que pueden provenir de los equipos de cómputo **Onyx**, **PC** o **MAC**; en ese sentido, también se puede añadir el uso de múltiples ventanas sobrepuestas, en las que se puede interactuar con las aplicaciones de los equipos de cómputo y otras fuentes de imagen como el **DVD** y **VHS**.

c.5.4 DESARROLLO

iXtli puede utilizarse como espacio de trabajo cuando se requiera el sistema de inmersión, interacción con los modelos y/o el despliegue amplio con múltiples imágenes. Con tal objetivo, en el **iXtli**, se tiene la capacidad de remover los nueve asientos delanteros e instalar una mesa de trabajo, donde un conjunto de **6** a **8** usuarios pueden debatir y/o analizar los datos proyectados en la pantalla.

c.5.5 CONFERENCIAS

En iXtli es posible realizar actividades académicas como cursos y conferencias, que utilicen la interacción con modelos tridimensionales, en conjunto con otros apoyos como animaciones, gráficas, páginas web, y otras, en un mismo espacio de trabajo. La flexibilidad de iXtli de visualizar al mismo tiempo diversas fuentes de video, proporciona herramientas para la docencia o difusión que hacen más accesible comunicar la información sobre los fenómenos estudiados o la difusión de proyectos de investigación.