UNIVERSIDAD AUTONOMA DE AGUASCALIENTES

CENTRO DE CIENCIAS BÁSICAS

DEPARTAMENTO DE SISTEMAS DE INFORMACIÓN

TESIS

Enseñanza de programación de estructuras de datos aplicando estrategias didácticas basadas en la teoría de carga cognitiva

PRESENTA

Ing. Martín Gabriel Romero Juárez

PARA OBTENER EL GRADO DE MAESTRO EN INFORMÁTICA Y TECNOLOGÍAS COMPUTACIONALES

COMITÉ TUTORAL

Tutor Dr. Carlos Argelio Arévalo Mercado

Cotutor Dra. Estela Lizbeth Muñoz Andrade

Asesor Dra. María Dolores Torres Soto

Aguascalientes, Ags. Julio 2020

DICTAMEN DE LIBERACION ACADEMICA PARA INICIAR LOS TRAMITES DEL EXAMEN DE GRADO

Fecha de dictaminación dd/mm/aa: **01/07/20**

NOMBRE: Martín Gabriel Romero Juárez **ID** 125886

PROGRAMA: Maestría en Informática y Tecnologías Computacionales **LGAC (del posgrado):** Tecnologías de Software y Objetos de Aprendizaje

TIPO DE TRABAJO: (X) Tesis ( ) Trabajo práctico

TITULO: **Enseñanza de programación de estructuras de datos aplicando estrategias didácticas basadas en la teoría de carga cognitiva.**

IMPACTO SOCIAL (señalar el impacto logrado): *Por medio de la mejora demostrada en el proceso de aprendizaje de estructuras de datos, se puede lograr una reducción en los porcentajes de aprobación de dicha asignatura y a su vez una mejora en la implementación de tecnologías derivadas de ella, en los futuros profesionistas*

INDICAR SI/NO SEGÚN CORRESPONDA:

Elementos para la revisión académica del trabajo de tesis o trabajo práctico:

SI	El trabajo es congruente con las LGAC del programa de posgrado
SI	La problemática fue abordada desde un enfoque multidisciplinario
SI	Existe *coherencia, continuidad y orden lógico del tema central con cada apartado*
SI	Los resultados del trabajo dan respuesta a las preguntas de investigación o a la problemática que aborda
SI	Los resultados presentados en el trabajo son de gran relevancia científica, tecnologica o profesional según el área
SI	El trabajo demuestra más de una aportación original al conocimiento de su área
SI	Las aportaciones responden a los problemas prioritarios del país
SI	Generó transferecia del conocimiento o tecnológica
SI	Cumpe con la ética para la investigación (reporte de la herramienta antiplagio)

El egresado cumple con lo siguiente:

SI	Cumple con lo señalado por el Reglamento General de Docencia
SI	Cumple con los requisitos señalados en el plan de estudios (créditos curriculares, optativos, actividades complementarias,estancia, predoctoral, etc)
SI	Cuenta con los votos aprobatorios del comité tutoral, en caso de los posgrados profesionales si tiene solo tutorpodrá liberar solo el tutor
NA	*Cuenta con la carta de satisfacción del Usuario*
SI	Coincide con el título y objetivo registrado
SI	Tiene congruencia con cuerpos académicos
SI	Tiene el CVU del Conacyt actualizado
NA	Tiene el artículo aceptado o publicado y cumple con los requisitos institucionales (en caso que proceda)
En caso de Tesis por artículos científicos publicados	
NA	Aceptación o Publicación de los artículos según el nivel del programa
NA	El estudiante es el primer autor
NA	El autor de correspondencia es el Tutor del Núcleo Académico Básico
NA	En los artículos se ven reflejados los objetivos de la tesis, ya que son producto de este trabao de investigación.
NA	Los artículos integran los capítulos de la tesis y se presentan en el idioma en que fueron publicados
NA	La aceptación o publicación de los artículos en revistas indexadas de alto impacto

Con base a estos criterios, se autoriza se continúen con los trámites de titulación y programación del examen de grado

Si X

No

FIRMAS

Elaboró:

* NOMBRE Y FIRMA DEL CONSEJERO SEGÚN LA LGAC DE ADSCRIPCION: Dr. Cesar Eduardo Velázquez Amador

NOMBRE Y FIRMA DEL SECRETARIO TÉCNICO: M.I.C. Jorge Eduardo Macías Luévano

*** En caso de conflicto de intereses, firmará un revisor miembro del NAB de la LGAC correspondiente distinto al tutor o miembro del comité tutoral, asignado por el Decano**

Revisó:

NOMBRE Y FIRMA DEL SECRETARIO DE INVESTIGACIÓN Y POSGRADO: Dra. en C. Bio. Haydée Martínez Ruvalcaba

Autorizó:

NOMBRE Y FIRMA DEL DECANO: M. en C. Jorge Martín Alférez Chávez

Nota: procede el trámite para el Depto. de Apoyo al Posgrado

En cumplimiento con el Art. 105C del Reglamento General de Docencia que a la letra señala entre las funciones del Consejo Académico: .... Cuidar la eficiencia terminal del programa de posgrado y el Art. 105F las funciones del Secretario *Técnico, llevar el seguimiento de los alumnos.*

UNIVERSIDAD AUTONOMA DE AGUASCALIENTES

CARTA DE VOTO APROBATORIO
INDIVIDUAL

M. en C. Jorge Martín Alférez Chávez
DECANO (A) DEL CENTRO DE CIENCIAS BASICASS

P R E S E N T E

Por medio del presente como *TUTOR* designado del estudiante *MARTÍN GABRIEL ROMERO JUÁREZ* con ID *125886* quien realizó la *tesis* titulada: ***ENSEÑANZA DE PROGRAMACIÓN DE ESTRUCTURAS DE DATOS APLICANDO ESTRATEGIAS DIDÁCTICAS BASADAS EN LA TEORÍA DE CARGA COGNITIVA***, un trabajo propio, innovador, relevante e inédito y con fundamento en el Artículo 175, Apartado II del Reglamento General de Docencia doy mi consentimiento de que la versión final del documento ha sido revisada y las correcciones se han incorporado apropiadamente, por lo que me permito emitir el **VOTO APROBATORIO**, para que *él* pueda proceder a imprimir*la* así como continuar con el procedimiento administrativo para la obtención del grado.

Pongo lo anterior a su digna consideración y sin otro particular por el momento, me permito enviarle un cordial saludo.

A T E N T A M E N T E
"Se Lumen Proferre"
Aguascalientes, Ags., a *01* **de** *julio* **de** *2020.*

Dr. Carlos Argelio Arreola Mercado
Tutor de *tesis*
El *nombre completo que aparece en el Voto Aprobatorio debe*
coincidir con el que aparece en el documento empastado. No se puede abreviar, ni omitir nombres

c.c.p.- Interesado
c.c.p.- Secretaría Técnica del Programa de Posgrado

UNIVERSIDAD AUTONOMA DE AGUASCALIENTES

CARTA DE VOTO APROBATORIO
INDIVIDUAL

M. en C. Jorge Martín Alférez Chávez
DECANO (A) DEL CENTRO DE CIENCIAS BASICASS
_____
P R E S E N T E

Por medio del presente como *CO-TUTORA* designada del estudiante *MARTÍN GABRIEL ROMERO JUÁREZ* con ID *125886* quien realizó la *tesis* titulada: *ENSEÑANZA DE PROGRAMACIÓN DE ESTRUCTURAS DE DATOS APLICANDO ESTRATEGIAS DIDÁCTICAS BASADAS EN LA TEORÍA DE CARGA COGNITIVA*, un trabajo propio, innovador, relevante e inédito y con fundamento en el Artículo 175, Apartado II del Reglamento General de Docencia doy mi consentimiento de que la versión final del documento ha sido revisada y las correcciones se han incorporado apropiadamente, por lo que me permito emitir el **VOTO APROBATORIO**, para que *él* pueda proceder a imprimir*la* así como continuar con el procedimiento administrativo para la obtención del grado.

Pongo lo anterior a su digna consideración y sin otro particular por el momento, me permito enviarle un cordial saludo.

A T E N T A M E N T E
"Se Lumen Proferre"
Aguascalientes, Ags., a *01* **de** *julio* **de** *2020.*

Dra. Estela Lizbeth Muñoz Andrade
Co-Tutor de *tesis*
El *nombre completo que aparece en el Voto Aprobatorio debe*
coincidir con el que aparece en el documento empastado. No se puede abreviar, ni omitir nombres

c.c.p.- Interesado
c.c.p.- Secretaría Técnica del Programa de Posgrado

UNIVERSIDAD AUTONOMA DE AGUASCALIENTES

CARTA DE VOTO APROBATORIO
INDIVIDUAL

M. en C. Jorge Martín Alférez Chávez
DECANO (A) DEL CENTRO DE CIENCIAS BASICASS

P R E S E N T E

Por medio del presente como *ASESORA* designada del estudiante *MARTÍN GABRIEL ROMERO JUÁREZ* con ID *125886* quien realizó la *tesis* titulada: *ENSEÑANZA DE PROGRAMACIÓN DE ESTRUCTURAS DE DATOS APLICANDO ESTRATEGIAS DIDÁCTICAS BASADAS EN LA TEORÍA DE CARGA COGNITIVA*, un trabajo propio, innovador, relevante e inédito y con fundamento en el Artículo 175, Apartado II del Reglamento General de Docencia doy mi consentimiento de que la versión final del documento ha sido revisada y las correcciones se han incorporado apropiadamente, por lo que me permito emitir el **VOTO APROBATORIO**, para que *él* pueda proceder a imprimir*la* así como continuar con el procedimiento administrativo para la obtención del grado.

Pongo lo anterior a su digna consideración y sin otro particular por el momento, me permito enviarle un cordial saludo.

A T E N T A M E N T E
"Se Lumen Proferre"
Aguascalientes, Ags., a *01* **de** *julio* **de** *2020.*

Dra. Maria Dolores Torres Soto
Asesora de *tesis*
El *nombre completo que aparece en el Voto Aprobatorio debe*
coincidir con el que aparece en el documento empastado. No se puede abreviar, ni omitir nombres

c.c.p.- Interesado
c.c.p.- Secretaría Técnica del Programa de Posgrado

AGRADECIMIENTOS

A CONACYT por brindarme el apoyo económico a lo largo de esta maestría para poder realizar este posgrado.

A la Universidad Autónoma de Aguascalientes por permitirme estudiar un posgrado de calidad, contando con una educación de excelencia, con profesores altamente capacitados e infraestructura de primer nivel.

A mi comité tutorar, el Dr. Carlos Argelio Arévalo Mercado, la Dra. Estela Lizbeth Muñoz Andrade y la Dra. María Dolores Torres Soto, por todo el apoyo brindado y por su valioso tiempo para la edición y mejora de este trabajo de investigación. Y al Maestro Jorge Eduardo Macías Luevano por el apoyo brindado a lo largo de esta maestría.

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

El presente trabajo de investigación tiene como objetivo diseñar e implementar material didáctico basado en la teoría de la carga cognitiva, particularmente usando el efecto de ejemplos resueltos, en combinación con la identificación de planes de programación, para la materia de estructura de datos con el objetivo de medir la efectividad de dicho material en el aprendizaje de los alumnos que estudian materias afines a las ciencias computacionales de la Universidad Autónoma de Aguascalientes.

Las bases teóricas de la investigación son: Teoría de la Carga Cognitiva, Efecto de ejemplo resuelto, Efecto de atención dividida y Planes de programación.

La teoría de la carga cognitiva (Sweller et al., 2019) fue desarrollada como una teoría de diseño instruccional basada en aspectos de la arquitectura cognitiva humana. Teniendo como componentes básicos, la memoria de trabajo, la memoria a largo plazo y su relación entre ellas. La capacidad y duración se vuelve limitadas en la memoria de trabajo cuando tiene que procesar información nueva, pero estas limitaciones desaparecen cuando la memoria de trabajo procesa información que previamente fue almacenada en la memoria de largo plazo. La memoria de largo plazo es virtualmente ilimitada en capacidad, almacenando la información en forma de esquemas.

Los ejemplos resueltos se centran en los diferentes estados de un problema y en los pasos de la solución, lo que permite a los alumnos inducir soluciones o esquemas generalizados (Sweller et al., 1998). Por lo que estudiar utilizando ejemplos resueltos puede facilitar la construcción de esquemas y transferir el

proceso de resolver el problema más que realmente resolver los problemas equivalentes.

(Van Merriënboer, 1990) define los planes de programación como:

“Secuencias estereotípicas de instrucciones de computadora o plantillas de lenguaje de programación. Esas plantillas pueden estar relacionados con una línea de programa (por ejemplo., PRINT “text”; variable), varias líneas de programa (por ejemplo., una estructura de bucle con inicializaciones adecuadas sobre el bucle y contadores o totales acumulados colocados correctamente dentro del bucle) o programas completos (por ejemplo., una plantilla general de entrada-proceso-salida). Por lo tanto, mientras que las plantillas de bajo nivel proporcionan declaraciones para usar en una situación particular, las plantillas de alto nivel son aplicables en una variedad cada vez más amplia de situaciones.”

Los planes de programación son plantillas de código, las cuales pueden ser una línea de código (la declaración de una variable, o la operación entre dos variables), varias líneas de código (una estructura de bucle, una estructura condicional) o programas completos (una plantilla de entrada-proceso-salida). Teniendo plantillas con niveles de complejidad variados.

Para la medición de la efectividad de la solución desarrollada, se llevó a cabo un cuasiexperimento con dos grupos (Control n = 35, Experimental n = 36) alumnos de 3° semestre de la carrera de Ingeniería en Sistemas Computacionales de la Universidad Autónoma de Aguascalientes. El análisis de diferencias de medias de muestras relacionadas (pre-post) del grupo experimental, mostró que existieron ganancias significativas en el aprendizaje de listas simplemente ligadas, sugiriendo que la herramienta desarrollada a partir de la teoría de la carga cognitiva tuvo un efecto positivo en el aprendizaje de los estudiantes.

ABSTRACT

This research aims to design and implement teaching material using the Worked Example, and Split Attention effects of Cognitive Load Theory, combining the identification of programming plans, all within the subject of data structures. Included in the objectives is the empirical measurement of the effectiveness of the material with subjects related to computer science at the Autonomous University of Aguascalientes.

Cognitive Load Theory was developed as an instructional design theory based on aspects of human cognitive architecture. It identifies as its basic components, working memory, long-term memory, and the relationship between them. Working Memory has limited capacity and duration but it´s the place where new information is processed. These limitations are reduced when Working memory retrieves information that was previously stored in the long-term memory. Long-term memory is said to be virtually unlimited in capacity, storing information in the form of schemas.

Worked examples focus on showing the different states and steps of a problem and its solution, allowing students to identify and store generalized solutions or schemas (Sweller et al., 1998). And so, it is theorized that using worked examples can make it easier to build Schemas and transfer the process of solving to long term memory rather than solving the equivalent problems with general problem-solving strategies.

Other theoretical aspect of this research where focused on what Van Merriënboer, (1990) defines as programming plans: Stereotypical sequences of computer instructions or programming language templates. Those templates can be related to one program line (e.g., PRINT "text"; variable), multiple program lines (e.g., a loop

structure with proper initializations to the loop, and counters or totals correctly placed inside of the loop) or complete programs (e.g., a general input-process-output template). Therefore, while low-level templates provide statements for use in a situation, high-level templates are applicable in an increasingly wide variety of situations. And so, programming plans are code templates, which can be one line of code (the declaration of a variable, or the operation between two variables), several lines of code (a loop structure, a conditional structure) or complete programs (an input-process-output template). Having templates with varying levels of complexity.

To measure the effectiveness of the developed software solution, a quasi-experiment was carried out with two groups (Control n = 35, Experimental n = 36) of 3rd semester students of the Computer Systems Engineering degree at the Autonomous University of Aguascalientes. The corresponding t test of related samples (pre-post) of the experimental group, showed that there were significant gains in learning simple linked lists, suggesting that the tool developed from the theory had a positive effect on student learning.

INTRODUCCIÓN

Antecedentes

El reciente desarrollo tecnológico ha llevado a una situación en la que muchos de nuestros niños trabajarán en una profesión que aún no existe, y trabajarán en la solución de problemas que aún no se han materializado. Para prepararlos para esto, necesitamos cambiar nuestro enfoque a la enseñanza (Hromkovič & Lacher, 2017).

Hromkovič & Lacher (2017) hacen una reflexión sobre la enseñanza: *"No enseñe los productos finales de la ciencia, la tecnología y las humanidades, y no considere que el objetivo más alto es capacitarlos para aplicarlos con éxito. Para el último conocimiento puede encontrarse desactualizado con el tiempo. Enseñe el proceso de descubrir nuevos conocimientos, enseñe la necesidad de buscar nuevas soluciones, enseñe las formas de recopilar experiencias y formule hipótesis, enseñe las formas de verificar hipótesis, enseñe cómo otros pueden estar convencidos de la verdad descubierta, enseñe la forma constructiva de pensar en crear nuevos productos y finalmente nuevas tecnologías, y enseñe los procesos de prueba y mejora de los productos de nuestro trabajo."*

La enseñanza de lenguajes de programación es uno de los fundamentos de la educación en informática y, por lo general, uno de los primeros cursos que toman los estudiantes novatos. Es importante que los maestros capten la atención de los estudiantes y fortalezcan su motivación para aprender a programar con la ayuda de una variedad de métodos de enseñanza. El aprendizaje de la programación es una tarea difícil para muchos estudiantes, debido a que es una habilidad de múltiples capas que es aburrida, intimidante y no está relacionada con la experiencia cotidiana en la que los estudiantes solo aprendieron en un contexto único (Rosminah et al., 2012) por lo que los maestros buscan nuevos métodos para

facilitar el aprendizaje. A lo largo de los años, los maestros han discutido la mejor manera de enseñar a los programadores novatos, qué enseñar y cómo, estilos de aprendizaje y herramientas de aprendizaje, cómo los estudiantes pueden estar motivados y apoyados y qué lenguajes de programación deben enseñarse (Matthíasdóttir, 2006).

Para los estudiantes que inician en la programación, crear un programa desde cero puede crearles frustración y disminuir su motivación por el aprendizaje. Además, un problema de programación puede ser resuelto de más de una forma. Ya que, para cualquier problema de programación, la solución encontrada por el alumno puede llevar bastante tiempo y ser diferente para cada estudiante (Chang et al., 2000).

Los estudiantes muestran un bajo rendimiento en los cursos de programación básica debido a que el aprendizaje de la programación impone una gran demanda cognitiva y de conocimiento en los novatos. Debido al contenido que abarcan los cursos, por ejemplo, el conocimiento de un lenguaje de programación específico, la comprensión de conceptos y desarrollo de programación básicos que incluyen variables, bucles, condiciones, abstracción y procedimientos. Cualquier idea errónea que el estudiante tenga sobre estos conceptos puede resultar en dificultades de programación (Rahimi et al., 2017).

Cuando se observa el desempeño de un curso introductorio de primer año en programación en un plan de estudios no vocacional, se pueden detectar diferentes problemáticas: dificultades de aprendizaje, bajo rendimiento, alto abandono escolar. Muchas instituciones en las últimas décadas han propuesto y desarrollado proyectos experimentales para la introducción de informática y tecnología digital en escuelas (Solitro et al., 2017).

Aprender a programar es difícil. Los programadores novatos sufren una amplia gama de dificultades y déficits. Los cursos de programación generalmente se consideran difíciles y, a menudo, tienen las tasas de abandono más altas (Robins et al., 2003).

Aprendizaje de la programación

La premisa por la cual se considera la enseñanza de la programación como un medio para lograr un mejor rendimiento escolar es que a través de un proceso de inducción es posible desarrollar un progreso cognitivo. Este proceso comienza observando ejemplos o problemas específicos, encontrando diferentes soluciones y haciendo generalizaciones. Si se aborda de esta manera, los estudiantes en un nivel más avanzado aprenderán a hacer deducciones. La experiencia o el conocimiento que adquieren los niños debe ser muy específico y permitirles adquirir el material paso a paso. Por lo tanto, los niños pasan por la experiencia de sugerir y crear soluciones que luego se pueden poner a prueba de inmediato. Al programar, el alumno se da cuenta de lo que significa internalizar, automatizar, ensamblar y reutilizar instrucciones de código en programas. La eficiencia y la complejidad de una solución les enseñará automáticamente a pensar algorítmicamente (Herrera Loyo, 2018).

Aprender a programar no es una tarea sencilla, (Du Boulay, 1986) menciona que las dificultades de aprender a programar pueden ser divididas en cinco áreas. Primera área, tiene que ver con la orientación del problema, que problema se está tratando de resolver con la programación. Segunda área, son las dificultades asociadas con entender las propiedades generales de la máquina, la cual se está aprendiendo a controlar. Tercera área, son los problemas asociados con la notación de los lenguajes que el programador debe de aprender y dominar, tanto la sintaxis como la semántica del lenguaje. Cuarta área, todas las dificultades asociadas con la adquisición de estructuras estándar o planes, como por ejemplo los ciclos. Y por

último la quinta área, aquí el problema es que los estudiantes deben de dominar la pragmática de la programación, es decir, que el alumno debe de aprender las habilidades de especificar, desarrollar, probar y depurar el programa desarrollado usando las herramientas que tenga disponibles.

Al resolver un problema de programación (escribir un programa), los estudiantes encontrarán algún aspecto del problema que no entienden (un punto muerto) (Bonar & Soloway, 1985). Para salir del punto muerto donde se encuentran, los estudiantes buscan una forma de resolver el aspecto del problema que no entienden (un parche). A menudo la solución aplicada implica dejar de lado su conocimiento de los procedimientos paso a paso del lenguaje natural que serían aplicables en una situación similar, lo que genera un error.

(S. Garner, 2002) habla sobre lo difícil que es aprender a escribir programas computacionales; lo anterior, derivado del bajo aprovechamiento académico que tienen los estudiantes en los primeros cursos de programación. Debido a que los estudiantes deben de desarrollar tres habilidades al mismo tiempo: aprender a usar el entorno de desarrollo del programa; aprender la sintaxis del lenguaje de programación; y desarrollar el diseño lógico.

El interés por parte de los alumnos en aprender programación no es muy alto, ya que el contenido de los cursos se vuelve altamente lógico y abstracto, lo que causa que al inicio se tenga problemas con el aprendizaje y haga sentir al estudiante que el curso es difícil y pierda y el interés en aprender (Wang, 2012).

Enseñanza de estructuras de datos

Una de las razones comunes por las que los estudiantes no completan el curso de estructuras de datos se debe a que no concluyen las tareas de programación o plantean una solución de una manera deficiente. Dado que es fundamental para el éxito de los estudiantes que completen con éxito las tareas, los instructores tienen el reto de desarrollar tareas interesantes para los estudiantes (Lawrence, 2004).

El proceso de enseñanza - aprendizaje de la asignatura Estructura de Datos en las carreras del área computacional, ha presentado problemas con la asimilación, análisis y creación de habilidades en el alumno que les permiten el diseño adecuado, eficiente y óptimo, de las estructuras de datos para representar la información y las operaciones básicas que con ellas se realizan, principalmente con el tema de listas enlazadas.

Los libros de estructuras de datos normalmente usan lenguajes como C, C++, JAVA u otros lenguajes de programación. (Wang, 2012) menciona que muchos de los estudiantes no comprenden la programación en lenguaje C de manera sólida, lo que lleva a que no sepan aplicar conceptos como estructuras, punteros y otros tipos de datos, temas que son enseñados en los cursos de estructuras de datos, lo que genera miedo en los estudiantes desde el inicio.

Para el caso de las estructuras de datos, el estudiante debe centrar su atención en la manera de cómo organizar un conjunto de información para poder responder de manera rápida y clara a las dudas que se presenten (Osvaldo Cairó & Silvia Guardati, 2006).

La visualización y/o animación de las estructuras de datos pueden ayudar a introducir los conceptos en la memoria a largo plazo asegurando una cobertura adecuada de los conceptos. Este enfoque permite centrarse simultáneamente, tanto en composición como en la exploración de los estudiantes. Incluso con la presión del tiempo que se tenga para aprender, los estudiantes pueden hacer malabares con el aprendizaje aplicando sus propios métodos mentales para producir conocimiento (Almeida et al., 2004; Segura Díaz & Pita Andreu, 2007).

En la literatura se han encontrado estudios que diversos autores han realizado buscando dar solución al problema de la enseñanza de estructura de datos. Hasta el momento existen varias propuestas, pero no existe alguna en la que se haga uso de un sistema multimedia - interactivo específicamente diseñado para enseñar el tema de árboles binarios.

Problemática de reprobación en la Universidad Autónoma de Aguascalientes

En el Centro de Ciencias Básicas de la Universidad Autónoma de Aguascalientes (UAA), existen tres programas de licenciatura afines a las ciencias computacionales: la Licenciatura en informática y Tecnologías Computacionales (LITC), la Ingeniería en Sistemas Computacionales (ISC) y la Ingeniería en Computación Inteligente (ICI), de las cuales se realizan informes semestrales de índices de reprobación por materia e índices de deserción por semestre, como se muestra a continuación en las cifras generadas por el Departamento de Estadística Institucional de la UAA. (Universidad Autónoma de Aguascalientes, 2019).

Los tres programas de licenciatura cuentan con asignaturas introductorias al aprendizaje de la programación en los dos primeros semestres y con asignaturas de Estructuras de Datos en sus planes de estudio, entre 3er y 4to semestre.

En el caso del programa de Licenciatura en Informática y Tecnologías Computacionales (LITC) (ver Tabla 1. Índices de reprobación Licenciatura en Informática y Tecnologías Computacionales) la asignatura de "Algoritmos Computacionales" muestra en promedio un porcentaje de reprobación del 23%, entre 2016 y 2018, teniendo la cifra más alta el 45.65%. La materia de estructura de datos muestra una reprobación del 20.49% en promedio entre 2016 y 2018. Las materias de programación estructurada muestran un promedio de reprobación del 29.34% entre 2017 y 2018, respectivamente.

Tabla 1. Índices de reprobación Licenciatura en Informática y Tecnologías Computacionales

	SEMESTRE	PRESENTARON	APROBACION		REPROBACION		REPROBACION POR CALIFICACION		REPROBACION POR OTRA RAZON	
AGOSTO-DICIEMBRE 2018										
ALGORITMOS COMPUTACIONALES	1	**46**	25	54.35%	**21**	**45.65%**	21	45.65%	0	0.00%
ESTRUCTURA DE DATOS	3	**42**	32	76.19%	**10**	**23.81%**	9	21.43%	1	2.38%
ENERO-JUNIO 2018										
PROGRAMACION ESTRUCTURADA	2	**46**	33	71.74%	**13**	**28.26%**	**5**	10.87%	**8**	17.39%
ESTRUCTURAS COMPUTACIONALES	2	**0**	0	0.00%	**2**	**100.00%**	1	50.00%	1	50.00%
PROGRAMACION ORIENTADA A OBJETOS	4	**26**	24	92.31%	**2**	**7.69%**	1	3.85%	1	3.85%
AGOSTO-DICIEMBRE 2017										
ALGORITMOS COMPUTACIONALES	1	**44**	40	90.91%	**4**	**9.09%**	0	0.00%	4	9.09%
ESTRUCTURA DE DATOS	3	**37**	30	81.08%	**7**	**18.92%**	5	13.51%	2	5.41%
ENERO-JUNIO 2017										
PROGRAMACION ESTRUCTURADA	2	**46**	32	69.57%	**14**	**30.43%**	**0**	0.00%	**14**	30.43%
PROGRAMACION ORIENTADA A OBJETOS	4	**39**	37	94.87%	**2**	**5.13%**	2	5.13%	0	0.00%
AGOSTO-DICIEMBRE 2016										
ALGORITMOS COMPUTACIONALES	1	**46**	39	84.78%	**7**	**15.22%**	5	10.87%	2	4.35%
ESTRUCTURA DE DATOS	3	**48**	39	81.25%	**9**	**18.75%**	7	14.58%	2	4.17%

Para el caso del programa de Ingeniería en Sistemas Computacionales (ISC) (ver Tabla 2), el porcentaje promedio de reprobación de la materia "Programación I" es del 34.65%, entre 2017 y 2018 (ver **¡Error! No se encuentra el origen de la referencia.**). Por su parte, los porcentajes de reprobación de la materia de estructuras de datos son más bajos, siendo el promedio entre 2016 y 2018 de 9.69%.

Tabla 2. Índices de reprobación Ingeniería en Sistemas Computacionales

	SEMESTRE	PRESENTARON	APROBACION		REPROBACION		REPROBACION POR CALIFICACION		REPROBACION POR OTRA RAZON	
AGOSTO-DICIEMBRE 2018										
ESTRUCTURAS DE DATOS	3	**116**	100	86.21%	**16**	**13.79%**	8	6.90%	8	6.90%
PROGRAMACION II	3	**129**	111	86.05%	**18**	**13.95%**	13	10.08%	5	3.88%
ENERO-JUNIO 2018										
PROGRAMACION I	2	**131**	83	63.36%	**48**	**36.64%**	45	34.35%	3	2.29%
PROGRAMACION III	4	**96**	91	94.79%	**5**	**5.21%**	3	3.13%	2	2.08%
AGOSTO-DICIEMBRE 2017										
ESTRUCTURAS DE DATOS	3	**111**	93	83.78%	**18**	**16.22%**	10	9.01%	8	7.21%
PROGRAMACION II	3	**117**	80	68.38%	**37**	**31.62%**	32	27.35%	5	4.27%
ENERO-JUNIO 2017										
PROGRAMACION I	2	**123**	78	63.41%	**45**	**36.59%**	43	34.96%	2	1.63%
PROGRAMACION III	4	**97**	94	96.91%	**3**	**3.09%**	1	1.03%	2	2.06%
AGOSTO-DICIEMBRE 2016										
ESTRUCTURAS DE DATOS	3	**114**	95	83.33%	**19**	**16.67%**	15	13.16%	4	3.51%
PROGRAMACION II	3	**107**	87	81.31%	**20**	**18.69%**	15	14.02%	5	4.67%

Finalmente, el programa de Ingeniería en Computación Inteligente (ICI) (ver Tabla 3), muestra porcentajes de reprobación significativamente menores a los otros dos programas, tanto en la asignatura de Lenguajes de Computación I (8.44% en promedio de 2016 a 2018), como en la de estructuras computacionales (3.19%). En el caso de este programa (ICI), existen variantes de curriculares en relacionadas con el tema de estructuras de datos, dado que esta rama de las ciencias computacionales se estudia en cuatro asignaturas diferentes, dado el perfil orientado a la inteligencia artificial de esta licenciatura.

Tabla 3. Índices de reprobación Ingeniería en Computación Inteligente

	SEMESTRE	PRESENTARON	APROBACION		REPROBACION		REPROBACION POR CALIFICACION		REPROBACION POR OTRA RAZON	
AGOSTO-DICIEMBRE 2018										
FUNDAMENTOS DE ESTRUCTURAS COMPUTACIONALES	1	**49**	47	95.92%	**2**	**4.08%**	2	4.08%	0	0.00%
ESTRUCTURAS COMPUTACIONALES AVANZADAS	3	**46**	42	91.30%	**4**	**8.70%**	4	8.70%	0	0.00%
LENGUAJES DE COMPUTACION I	1	**50**	47	94.00%	**3**	**6.00%**	2	4.00%	1	2.00%
LENGUAJES DE COMPUTACION III	3	**41**	39	95.12%	**2**	**4.88%**	2	4.88%	0	0.00%
LENGUAJES DE COMPUTACION IV	5	**40**	39	97.50%	**1**	**2.50%**	1	2.50%	0	0.00%
ENERO-JUNIO 2018										
ESTRUCTURAS COMPUTACIONALES III	4	**40**	39	97.50%	**1**	**2.50%**	0	0.00%	1	2.50%
ESTRUCTURAS COMPUTACIONALES	2	**45**	42	93.33%	**3**	**6.67%**	3	6.67%	0	0.00%
LENGUAJES DE COMPUTACION II	2	**43**	42	97.67%	**1**	**2.33%**	1	2.33%	0	0.00%
AGOSTO-DICIEMBRE 2017										
FUNDAMENTOS DE ESTRUCTURAS COMPUTACIONALES	1	**47**	44	93.62%	**3**	**6.38%**	2	4.26%	1	2.13%
ESTRUCTURAS COMPUTACIONALES II	3	**42**	37	88.10%	**5**	**11.90%**	2	4.76%	3	7.14%
LENGUAJES DE COMPUTACION I	1	**47**	42	89.36%	**5**	**10.64%**	4	8.51%	1	2.13%
LENGUAJES DE COMPUTACION III	3	**42**	39	92.86%	**3**	**7.14%**	1	2.38%	2	4.76%
LENGUAJES DE COMPUTACION IV	5	**34**	34	100.00%	**0**	**0.00%**	0	0.00%	0	0.00%
ENERO-JUNIO 2017										
ESTRUCTURAS COMPUTACIONALES III	4	**36**	35	97.22%	**1**	**2.78%**	0	0.00%	1	2.78%
LENGUAJES DE COMPUTACION II	2	**43**	43	100.00%	**0**	**0.00%**	0	0.00%	0	0.00%
AGOSTO-DICIEMBRE 2016										
ESTRUCTURAS COMPUTACIONALES I	1	**45**	45	100.00%	**0**	**0.00%**	0	0.00%	0	0.00%
ESTRUCTURAS COMPUTACIONALES II	3	**40**	40	100.00%	**0**	**0.00%**	0	0.00%	0	0.00%
LENGUAJES DE COMPUTACION I	1	**46**	42	91.30%	**4**	**8.70%**	3	6.52%	1	2.17%
LENGUAJES DE COMPUTACION III	3	**40**	40	100.00%	**0**	**0.00%**	0	0.00%	0	0.00%
LENGUAJES DE COMPUTACION IV	5	**25**	25	100.00%	**0**	**0.00%**	0	0.00%	0	0.00%

De tal suerte, puede observarse que -con excepción del programa de Computación Inteligente- las bases algorítmicas de programación, representan un porcentaje significativo de reprobación para los alumnos de las carreras de ISC y LITC lo cual puede tener un impacto importante en la comprensión e implementación de estructuras datos y todas las aplicaciones que se derivan de ella.

Estado del Arte

Dentro del área de las ciencias cognitivas, existen importantes contribuciones que describen (y en algunos casos predicen) el comportamiento de la memoria humana, en actividades de aprendizaje y solución de problemas. Importantes contribuciones se han difundido en los últimos años en relación con la forma en que como el cerebro humano almacena y recupera información de corto y largo plazo, para identificar patrones y resolver problemas, encontrándose diferencias de comportamiento cognitivo entre lo que se consideran “expertos” y “novatos” en diferentes especialidades.

Ejemplos de algunas de estas teorías son la Teoría de Codificación Dual (Alty, 2002) la cual señala que el cerebro humano es capaz de almacenar “pares de datos” simultáneamente, que corresponden a imágenes y textos o sonidos y textos, lo cual representa ventajas desde el punto de vista instruccional. También, la teoría de los esquemas (Cooper & Sweller, 1987; Niesser, 1976) es ampliamente referenciada ya que describe como un conjunto de elementos individuales pueden ser almacenados como una sola unidad en la memoria de largo plazo, por medio de la repetición y la identificación de patrones. De tal suerte, un experto puede identificar complejos patrones como una unidad, incluso si los componentes discretos son numerosos. Existen ejemplos en el ajedrez (Chase & Simon, 1973; Egan & Schwartz, 1979) y en las matemáticas (Cooper & Sweller, 1987), que demuestran la existencia del almacenamiento de esquemas en la memoria de largo plazo del cerebro humano. A este fenómeno se le describe en idioma inglés como *“Chunking*” o “agrupamiento”.

Finalmente, y tomando partes de las anteriores teorías, la Teoría de la Carga Cognitiva (Sweller, 1994) ha cobrado especial relevancia en el estudio del aprendizaje y en el diseño de materiales educativos, por su respaldo empírico y por el carácter predictivo de la misma. La teoría de carga cognitiva se ha enfocado al aspecto instruccional y al diseño de materiales, sugiriendo principios de diseño con bases empíricas (Connolly et al., 2009; Kirschner et al., 2006; Sweller & Cooper, 1985), que predicen un conjunto de “efectos” (positivos y negativos) en la carga cognitiva de la memoria de corto y largo plazo y por lo tanto en el aprendizaje.

De tal suerte, para la presente tesis, se tomarán elementos de diseño de las ciencias cognitivas, para la elaboración de materiales de apoyo interactivos basados en software para el aprendizaje de las estructuras de datos.

CAPÍTULO I

PROBLEMA DE INVESTIGACIÓN

1.1. Objetivo general

Diseñar e implementar material didáctico basado en la teoría de la carga cognitiva, particularmente usando el ***efecto de ejemplo resuelto***, en combinación con la identificación de ***planes de programación***, para la materia de estructura de datos con el objetivo de medir la efectividad de dicho material en el aprendizaje de los alumnos que estudian materias afines a las ciencias computacionales de la Universidad Autónoma de Aguascalientes.

1.2. Objetivos específicos

- Delimitar la(s) temática(s) de las estructuras de datos sobre las cuales se enfocará el desarrollo del material basado en el efecto del problema por completar.
- Identificar los planes de programación utilizados en las temáticas de estructuras de datos seleccionadas.
- Identificar y evaluar el formato de diseño del material didáctico basado en el efecto de problemas por completar (escrito, código fuente y/o artefacto de software).
- Desarrollar material didáctico utilizando el formato de diseño seleccionado.
- Probar la eficacia del material didáctico diseñado en el aprendizaje de estructuras de datos, mediante pruebas controladas o semi-controladas.

1.3. Preguntas de investigación

1. ¿Cuáles serán las diferentes temáticas de estructuras de datos sobre las cuales se enfocará el desarrollo del material basado en el efecto del problema por completar?
2. ¿Cuáles son los planes de programación para las diferentes estructuras de datos sobre las cuales se enfocará el desarrollo del material basado en el efecto del problema por completar?
3. ¿Cuáles son las características del formato de diseño del material didáctico basado en el efecto de problemas por completar?
4. ¿Cómo diseñar el material didáctico utilizando el formato de diseño seleccionado de forma correcta?
5. ¿Qué tan eficiente es el material didáctico diseñado en el aprendizaje de estructuras de datos en las pruebas controladas o semi-controladas?

1.4. Justificación

En el campo de tratamiento de datos, la implementación de estructuras de datos cobra especial importancia dado su amplio espectro de uso en diversas aplicaciones computacionales, tales como el aprendizaje automático, los problemas de optimización y búsqueda, las tecnologías basadas en *Blockchain* y algunas variantes del *Big Data*. Sin embargo, como ya se ha mencionado, la enseñanza en el aula de estos temas tiene un alto grado de dificultad por la complejidad de los modelos mentales y computacionales requeridos para su programación.

En este sentido, en el ámbito académico se han desarrollado herramientas de visualización del comportamiento de las estructuras de datos básicas, tales como las pilas, colas, listas y árboles, orientadas a crear modelos mentales y mejorar la comprensión en los estudiantes sobre el comportamiento de dichas estructuras (Almeida et al., 2004; Segura Díaz & Pita Andreu, 2007). Aunque estas aplicaciones

son de especial utilidad para la comprensión a un alto nivel de abstracción de las estructuras de datos, suelen ser insuficientes para que los alumnos de ciencias computacionales puedan programar los algoritmos requeridos para su implementación. En otras palabras, las herramientas de visualización permiten ver mediante animaciones la ejecución de una estructura de datos, pero no apoyan en estrategias que indiquen como programarla.

De tal suerte, la presente tesis buscará proporcionar un apoyo didáctico en la forma de una aplicación Web, dirigida a alumnos del área de las ciencias computacionales que se encuentren estudiando un curso de estructuras de datos, que complemente la adquisición de estrategias de programación de estructuras de datos y tentativamente reduzca los índices de reprobación de tales cursos. Para asegurar su efectividad, tal herramienta estaría desarrollada bajo principios de diseño instruccional basados a su vez en ciencias cognitivas.

CAPÍTULO II

MARCO TEÓRICO

Los altos índices de reprobación y deserción en materias de programación han hecho que investigadores busquen las causas y soluciones a estos problemas. Es de vital importancia encontrar las causas que llevan a los estudiantes a tener un bajo desempeño en el aprendizaje de la programación para encontrar soluciones y métodos alternativos de aprendizaje que puedan ser implementados y asistir a los estudiantes mientras estudian programación (Tan et al., 2009).

2.1. Teoría de Carga Cognitiva

La teoría de la carga cognitiva (Sweller et al., 1998) es una teoría de aprendizaje e instrucción, su desarrollo se remonta a principios de la década de 1980 y proporcionó material instruccional que difería de las prácticas de enseñanza predominantes de la época (van Merriënboer & Sweller, 2005), y la cual describe las implicaciones del diseño instruccional basado en el modelo de la arquitectura cognitiva humana (Kalyuga, 2011). Una teoría que hace énfasis en las limitaciones de la memoria de trabajo como principales causantes de la efectividad del diseño instruccional (Sweller et al., 1998) y una memoria de largo plazo virtualmente ilimitada que almacena trozos de información representados por esquemas (Gerjets et al., 2009) (Paas et al., 2004).

La teoría de la carga cognitiva se centra en el aprendizaje de tareas cognitivas complejas, en las que los alumnos se ven abrumados por la cantidad de elementos de información interactivos que deben de procesar al mismo tiempo antes de que se realice un aprendizaje significativo. Por lo que el enforque de la teoría de la carga cognitiva es el control instruccional de la carga excesivamente alta impuesta sobre

tareas complejas. Para llevar a cabo este control, la teoría de la carga cognitiva utiliza el conocimiento actual de la arquitectura cognitiva humana para generar técnicas de instrucción (Paas et al., 2010).

Le teoría de la carga cognitiva soporta modelos explícitos de instrucción que ayudan a los maestros a enseñar de forma clara a los estudiantes, indicándoles que hacer y cómo hacerlo, en vez de que los estudiantes descubran la información ellos mismos (New South Wales. Centre for Education Statistics and Evaluation, 2017).

Como se mencionó anteriormente, la teoría de la carga cognitiva está centrada en la arquitectura cognitiva humana, en especial en las limitaciones de la memoria de trabajo (Leppink et al., 2014). La memoria de trabajo puede asemejarse a la conciencia. Las personas pueden ser consientes solo del contenido de la memoria de trabajo. Tiene varias limitaciones que son reconocidas y ampliamente aceptadas (Sweller et al., 1998). Solo es capaz de retener cerca de siete elementos de información a la vez; Cabe señalar que un elemento se define como cualquier material que necesite ser aprendido por el estudiante (Sweller, 1994). En cambio, cuando se requiere procesar información el número de elementos con los que puede lidiar disminuye a dos o tres elementos simultáneamente. Cualquier interacción entre los elementos mantenidos en la memoria de trabajo requerirá capacidad de la memoria de trabajo, lo que hará que se reduzca el número de elementos que puede soportar a la vez.

La memoria de trabajo puede ser afectada por la carga cognitiva intrínseca y la carga cognitiva extraña (Sweller, 1994). Las implicaciones de las limitaciones de la memoria de trabajo sobre el diseño instruccional pueden ser sobreestimadas. Por lo que cualquier diseño instruccional que ignora las limitaciones de la memoria de trabajo es inevitablemente deficiente (Sweller et al., 1998).

Los humanos no somos conscientes de la información almacenada en la memoria de largo plazo, hasta que esta pasa a ser procesada por la memoria de trabajo. Acorde a la teoría de esquemas, el conocimiento es almacenado en la memoria de largo plazo en forma de esquemas. Una de las funciones que provee es el mecanismo de organización y almacenamiento de conocimiento. Un esquema clasifica los elementos de información de acuerdo con la forma en que se utilizarán. Y es a través de la construcción de un número creciente de esquemas cada vez más complejos mediante la combinación de elementos que consisten en esquemas de nivel inferior en esquemas de nivel superior que se desarrollan las habilidades. (Sweller et al., 1998).

2.2. Categorías de la Carga Cognitiva

La teoría de la carga cognitiva distingue entre tres tipos de carga cognitiva: intrínseca, extraña y pertinente (Paas et al., 2004).

La carga causada por la naturaleza intrínseca de la tarea de aprendizaje (carga cognitiva intrínseca) y la carga cognitiva causada por el formato de instrucción (carga cognitiva extraña) más que por las características intrínsecas del aprendizaje. Es decir, el material instruccional debe diseñarse de tal manera que no sea demasiado complejo (carga intrínseca), para así reducir la carga en la memoria de trabajo producida por procesos que no contribuyen al aprendizaje (carga extraña) y optimiza la carga resultante de los procesos que fomentan el aprendizaje (carga pertinente) (van Gog et al., 2010). No fue hasta la segunda mitad de la década de 1990, que la investigación de la teoría de la carga cognitiva se centraba exclusivamente en diseños de instrucción que pretendían disminuir la carga cognitiva extraña (Schnotz & Kürschner, 2007).

El nivel de la carga cognitiva intrínseca para una tarea y el nivel de conocimiento es determinado por la interactividad entre los elementos. La baja interactividad de los elementos permite a elementos individuales ser aprendidos con una referencia mínima a otros elementos y así imponer una carga baja en la memoria de trabajo. Al contrario de lo que ocurre con la baja interactividad de elementos, una alta interactividad entre elementos consiste en que los elementos que interactúan intensamente, haciendo que no se puedan aprender de forma aislada (Sweller, 2010).

La carga en la memoria de trabajo no solo es impuesta por la complejidad intrínseca del material instruccional que necesita ser aprendido. también puede generada por los procedimientos de instrucción que no son óptimos para el aprendizaje (Sweller, 2010).

(Sweller, 2010) indica que la interactividad de los elementos es la principal fuente de carga de memoria de trabajo subyacente a la carga cognitiva tanto extraña como intrínseca. Si la interactividad del elemento puede reducirse sin alterar lo aprendido, la carga es extraña

La carga cognitiva pertinente no constituye una fuente independiente de carga cognitiva. Simplemente se refiere a los recursos de memoria de trabajo disponibles para tratar la interactividad del elemento asociada con la carga cognitiva intrínseca (Sweller, 2010).

2.3. Efectos de la Carga Cognitiva

2.3.1. Efecto de ejemplo resuelto

El efecto del ejemplo resuelto (Andersen et al., 2016; Caspersen & Bennedsen, 2007; Sweller & Cooper, 1985; Tarmizi & Sweller, 1988) opera bajo el principio de que se pueden proveer al aprendiz esquemas de solución de problemas de manera más eficiente, si se le proporcionan a éste explicaciones detalladas “paso a paso”, con énfasis en secciones particulares del problema que deben llamar la sobre las características de la solución aplicada y que lo hacen diferente a otros problemas. Esto impone una carga cognitiva menor en la memoria de corto plazo que si se utilizaran estrategias de resolución general (means-ends analysis, por su término en inglés), al tiempo que toda la carga cognitiva intrínseca (es decir, inherente al problema) queda encapsulada en el ejemplo resuelto. La literatura reporta suficiente evidencia empírica sobre la superioridad de usar estrategias instruccionales en diversos ámbitos educativos, que toman en cuenta este efecto (refs), contra otros diseños tradicionales.

El ejemplo clásico proporcionado por los autores es una explicación paso a paso de la solución de una ecuación básica:

Ejemplo: despejar “a” de la siguiente ecuación

$$(a + b) / c = d$$

$$a + b = dc$$

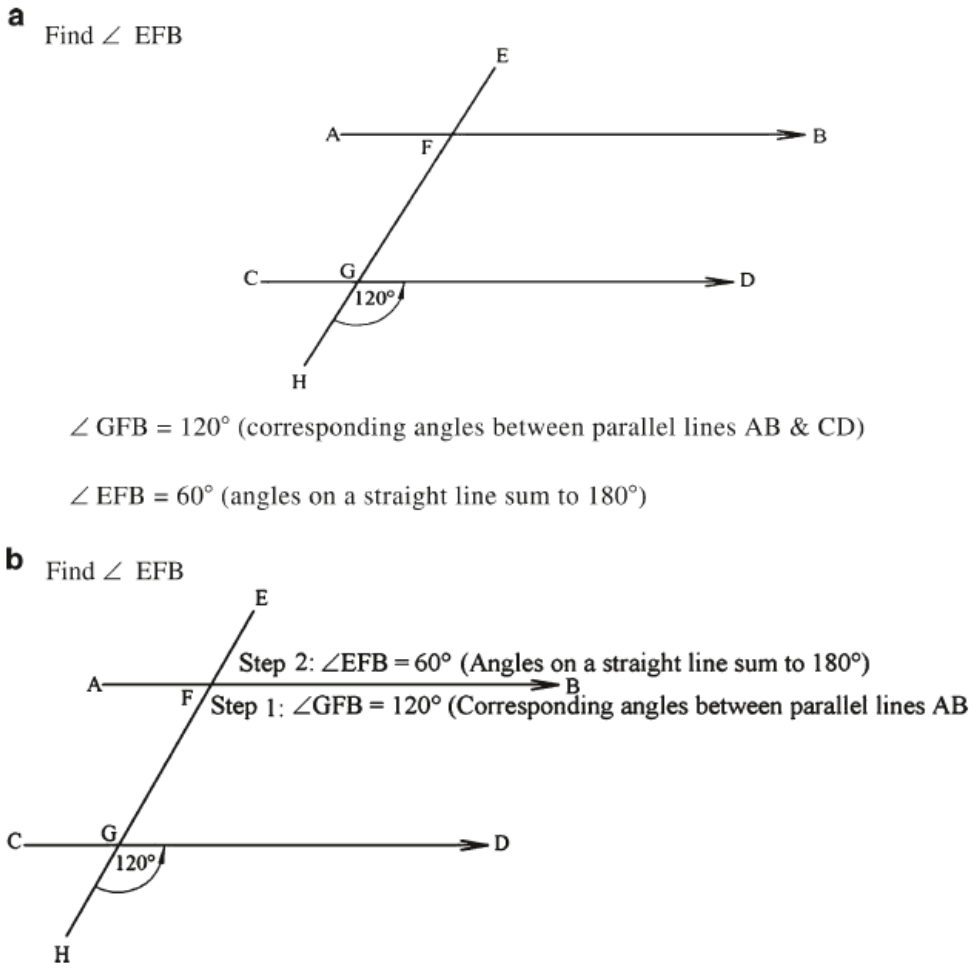
$$a = dc – b$$

2.3.2. Efecto de atención dividida

El efecto de atención divida establece que cuando se diseña material instruccional, incluyendo material multimedia, es importante evitar que el estudiante divida su

atención entre múltiples fuentes de información e integre mentalmente varias fuentes de información física o temporalmente dispares. (Ayres & Sweller, 2005).

Este efecto ocurre cuando se requiere que el aprendiz divida su atención entre dos fuentes de atención, ya sea separadas espacial o temporalmente. Sin embargo, de acuerdo con los autores, esta división no debe realizarse cuando los elementos a estudiar están "**altamente integrados**" y no pueden ser comprendidos por separado. Por ejemplo, en el área de la geometría, se propone que la descripción de los ángulos no se separe de las figuras correspondientes, en los correspondientes materiales didácticos (ver Figura 1).



En el ejemplo b se toma en cuenta el efecto de atención dividida, integrando la explicación textual al diagrama, en la solución del ejemplo resuelto.

Figura 1. Ejemplo de efecto de atención divida. (Tomado de (Sweller et al., 2011))

Para materiales multimedia, se propone que **la integración puede realizarse** en un ejemplo resuelto con el efecto de atención dividida, incluyendo una narrativa cuya secuencia guíe temporalmente al aprendiz (ya sea en forma de texto o sonidos) hacia el diagrama o esquema correspondiente.

2.4. Planes de programación

A lo largo del tiempo se ha hecho un llamado a las personas que trabajan en áreas afines a las ciencias computacionales y a las comunidades de Ingeniería de software a codificar plantillas estándar de programas. Las dos principales razones son: mejorar la fiabilidad y correcciones en el software, y mejorar la educación de los programadores (Rich, 2004).

Los programadores novatos usan estructuras esquematizadas llamadas planes. (Bonar & Soloway, 1985) propone dos tipos de planes y como se relacionan entre ellos.

El primer tipo de plan se conoce como conocimiento de programación de lenguaje natural paso a paso (conocimiento de preprogramación) y aporta a los programadores novatos conocimiento de las tareas estándar en los procedimientos de lenguaje natural paso a paso a su curso de programación introductoria. Las tareas incluyen bucles, tomar decisiones y especificar secuencias de acciones.

El segundo tipo de plan es conocimiento básico del lenguaje de programación a aprender (fragmentos de conocimiento de programación). Este tipo de plan representa el conocimiento que permite al estudiante escribir algunas partes del código de manera correcta. Además de enseñar la sintaxis y la semántica, muestra los objetivos y las tácticas críticas para implementar una tarea en un programa (en forma de código de programación).

Existen dos similitudes en particular entre los dos tipos de planes. Las similitudes funcionales, relacionadas con los ciclos, elección entre condiciones, contadores, entre otras, y las similitudes superficiales las cuales existen porque los lenguajes de

programación comparten muchas palabras con el lenguaje natural. Hay muchas entidades léxicas comunes en los dos conjuntos de planes, independientemente de la similitud funcional de los planes.

Las similitudes funcionales y de superficie entre conocimiento de preprogramación y fragmentos de conocimiento de programación permiten a un estudiante principiante en programación atravesar las dos diferentes estructuras de conocimiento.

(Adelson, 1981) llegó a la conclusión que los programadores ya sean expertos o novatos tienen categorías conceptuales para los elementos de un lenguaje de programación. En los programadores novatos, las categorías parecen ser de naturaleza sintáctica más que semántica y los elementos dentro de la categoría no están relacionados entre sí de manera fuertemente organizada. Esto puede cambiar conforme el programador gana experiencia, transformando la naturaleza de la categoría de sintáctica a semántica, volviéndose conceptualmente más complejos y teniendo una fuerte relación entre los elementos, lo que con lleva a que el programador pueda generar rutinas más complejas de código.

(Soloway & Ehrlich, 1984) sugieren que los programadores expertos cuentan con dos tipos de conocimiento en programación: los planes de programación, los cuales son fragmentos de código que representan acciones estereotipadas en un programa, y las reglas del discurso de programación, las reglas que especifican las convenciones en un programa, por ejemplo, que se nombre a una variable con respecto a la función que va a desempeñar. Ambos conocimientos juegan un papel importante al momento de comprender un programa.

(Van Merriënboer, 1990) define los planes de programación como:

“Secuencias estereotípicas de instrucciones de computadora o plantillas de lenguaje de programación. Esas plantillas pueden estar relacionados con una línea de programa (por ejemplo., PRINT “text”; variable), varias líneas de programa (por ejemplo., una estructura de bucle con inicializaciones adecuadas sobre el bucle y contadores o totales acumulados colocados correctamente dentro del bucle) o programas completos (por ejemplo., una plantilla general de entrada-proceso-salida). Por lo tanto, mientras que las plantillas de bajo nivel proporcionan declaraciones para usar en una situación particular, las plantillas de alto nivel son aplicables en una variedad cada vez más amplia de situaciones.”

Para los estudiantes que están en el área del diseño lógico, es necesario que aprendan los planes de programación fundamentales que se utilizan para crear programas (S. Garner, 2002).

CAPÍTULO III

METODOLOGÍA

3.1. Diseño de la solución

Se revisó el programa de la materia de estructuras de datos (ver anexo 1) que llevan los alumnos de Ingeniería en Sistemas Computacionales de 3º semestre con el objetivo de conocer las diversas estructuras de datos que son enseñadas en el curso.

Se optó por seleccionar el tema de listas ligadas, el cual se enseña en la unidad IV, debido a la complejidad del tema y los subtemas que son parte de las listas ligadas, esto con el propósito de poner generar un número suficiente de planes de programación.

Una vez seleccionado el tema de estructuras de datos con el que se planea trabajar, el cual fue listas ligadas, se revisó la literatura (Aguilar, 2006; Deitel & Deitel, 2004), para la identificación y el conocimiento a fondo todos los temas que son parte de las listas ligadas.

Se identificaron los temas fundamentales que los estudiantes deben de conocer y dominar para comprender e implementar los algoritmos de Listas Ligadas en lenguaje C. También se identificaron las categorías de las Listas Ligadas y las diferentes operaciones que se pueden aplicar a dichas listas.

Los temas que se identificaron y sin ellos no es posible llevar a cabo la implementación de listas ligadas en lenguaje C son: punteros, estructuras,

estructuras autoreferenciadas y el manejo de la memoria dinámica. Se identificaron las funciones correspondientes en lenguaje C para cada uno de estos temas.

Las listas ligadas se dividen en cuatro categorías, listas simplemente ligadas, listas doblemente ligadas, listas circulares simplemente ligadas, listas circulares doblemente ligadas. Cada una de estas listas funciona de manera diferente, pero las funciones principales que se pueden aplicar sobre estas estructuras de datos son las mismas: añadir, eliminar y búsqueda (Osvaldo Cairó & Silvia Guardati, 2006).

Para esta investigación se seleccionaron las listas simplemente ligadas debido a que son el tipo de listas ligadas más simple y el primer tipo de listas que se enseñan. Las operaciones que se pueden realizar sobre este tipo de listas son: insertar un nuevo nodo, eliminar un nodo, búsqueda de un nodo y recorrido de la lista. Como una función extra se agregó la edición del valor del miembro de un nodo. Además de las principales operaciones que se pueden realizar sobre las listas simplemente ligadas, existen subfunciones para insertar y eliminar un nodo.

Para la inserción de un nuevo nodo se pueden realizar las siguientes subfunciones: insertar un nuevo nodo al inicio o al final de la lista, también se puede insertar un nuevo nodo entre dos nodos.

La eliminación de un nodo en una lista simplemente ligada se puede llevar a cabo de la siguiente manera: eliminar el nodo inicial de la lista, eliminar el nodo final de la lista o eliminar un nodo intermedio.

Es importante identificar todos los temas que conforman las Listas Ligadas, las operaciones que se pueden aplicar sobre esta estructura de datos, con el fin de diseñar y crear de forma correcta los planes de programación.

3.1.1. Diseño y creación de planes de programación

Una vez identificados los diferentes temas que son abarcados por las Listas Simplemente Ligadas, se procedió a diseñar y crear los diversos planes de programación. Se crearon un total de cinco planes de programación (ver Tabla 4).

Tabla 4. Planes de programación

Número de plan de programación	Planes de programación
1	Declaración de variable tipo apuntador
2	Declaración de variable tipo estructura
3	Declaración de estructuras auto referenciadas
4	Gestión dinámica de memoria uso del operador **new**
5	Gestión dinámica de memoria uso del operador **delete**

Diseño y creación de ejemplos resueltos

Los ejemplos resueltos son diseñados en base a las funciones que se pueden implementar en las listas simplemente ligadas. Se identificaron un total de once ejemplos resueltos (ver Tabla 5).

Tabla 5. Ejemplos resueltos listas simplemente ligadas

Número de ejemplo resuelto	Ejemplos resueltos listas simplemente ligadas
1	Lista Simplemente Ligada insertar nodo inicial
2	Lista Simplemente Ligada insertar nodo inicio
3	Lista Simplemente Ligada insertar nodo final
4	Lista Simplemente Ligada insertar nodo antes de
5	Lista Simplemente Ligada insertar nodo después de
6	Lista Simplemente Ligada eliminar nodo inicio
7	Lista Simplemente Ligada eliminar nodo final
8	Lista Simplemente Ligada eliminar nodo con información x
9	Lista Simplemente Ligada buscar nodo
10	Lista Simplemente Ligada modificar información nodo
11	Lista Simplemente Ligada imprimir lista

A continuación, se describe cada uno de los temas, así como su respectivo plan de programación.

3.1.1.1. Apuntadores

Los apuntadores son uno de los temas más difíciles de dominar en C. Los apuntadores permiten realizar llamadas por referencia, crear y manipular estructuras de datos dinámicas, estructuras de datos que cambian su tamaño en tiempo de ejecución, como lo son: listas ligadas, pilas, colas y árboles.

Los valores que se almacenan en una variable de tipo apuntador son direcciones de memoria de variables que almacenan valores específicos. El nombre de una variable hace referencia directa a un valor, y un apuntador hace referencia indirecta a un valor.

Los apuntadores son la pieza clave de las listas ligadas, almacenando la dirección de memoria de los nodos que conforman las listas ligadas (ver Figura 2).

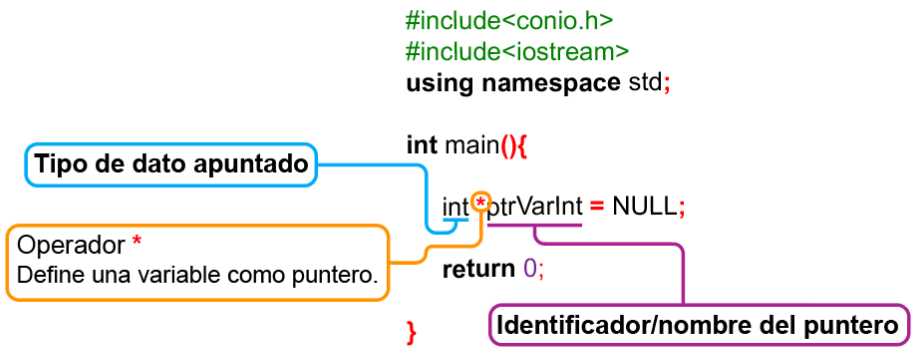


Figura 2. Plan de programación, Declaración de variable tipo apuntador

3.1.1.2. Estructuras

Las estructuras son una colección de variables, llamadas miembros, los cuales pueden ser de un tipo de dato diferentes, relacionadas bajo un nombre. Las estructuras deben de ser declaras antes de poder ser utilizadas. Pueden contener una cantidad ilimitada de miembros. Cada miembro tiene un nombre único, denominado nombre del miembro.

Comúnmente los elementos que conforman las listas ligadas son creados utilizando estructuras, más en específico, estructuras autoreferenciadas, y con el uso combinado de apuntadores se hace más fácil la creación de estructuras de datos más complejas, como listas ligadas, pilas, colas y árboles (ver Figura 3).

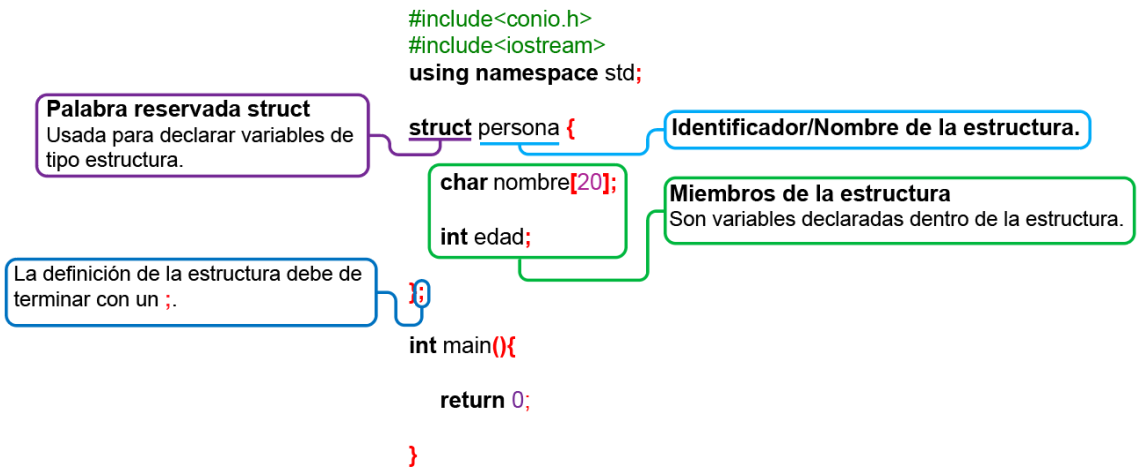


Figura 3. Plan de programación, Declaración de estructura

3.1.1.3. Estructuras autoreferenciadas

La diferencia entre estructura y estructura autoreferenciada es que esta última contiene un miembro apuntador, el cual guarda la dirección de memoria de una estructura del mismo tipo (ver Figura 4).

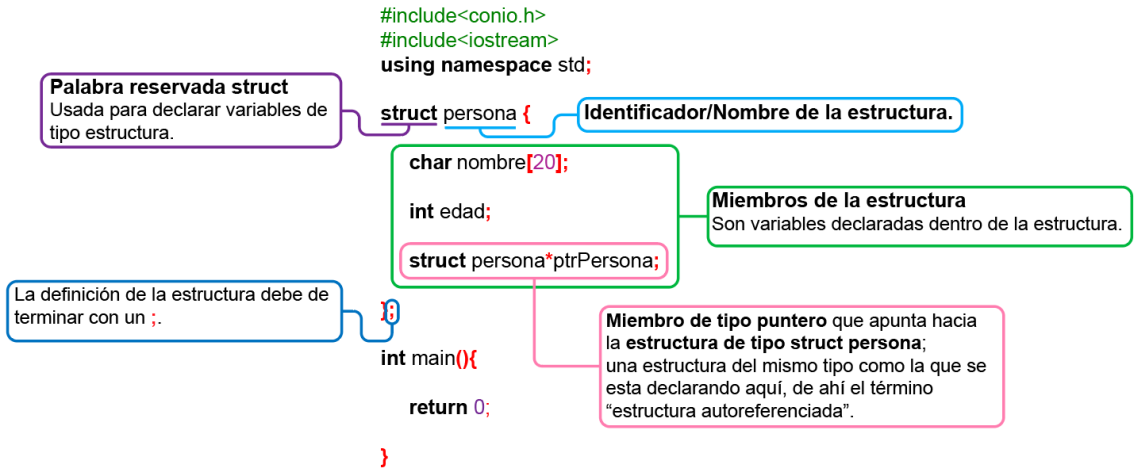


Figura 4. Plan de programación, Declaración de estructura autoreferenciada

3.1.1.4. Gestión dinámica de memoria

Existen ocasiones donde no se conoce la cantidad de memoria necesaria hasta el momento de la ejecución de un programa. La forma de resolver este inconveniente es a través del uso de apuntadores y técnicas de asignación dinámica de memoria, y así el programa puede crear y destruir la asignación dinámica en cualquier momento de la ejecución. El limite que se puede asignar de forma dinámica depende de la memoria física disponible en la computadora.

C++ dispone de dos operadores para la gestión de la memoria, ***new*** y ***delete*** (ver Figura 5 y Figura 6), que asignan y liberan la memoria, respectivamente, de una zona de memoria denominada almacén libre.

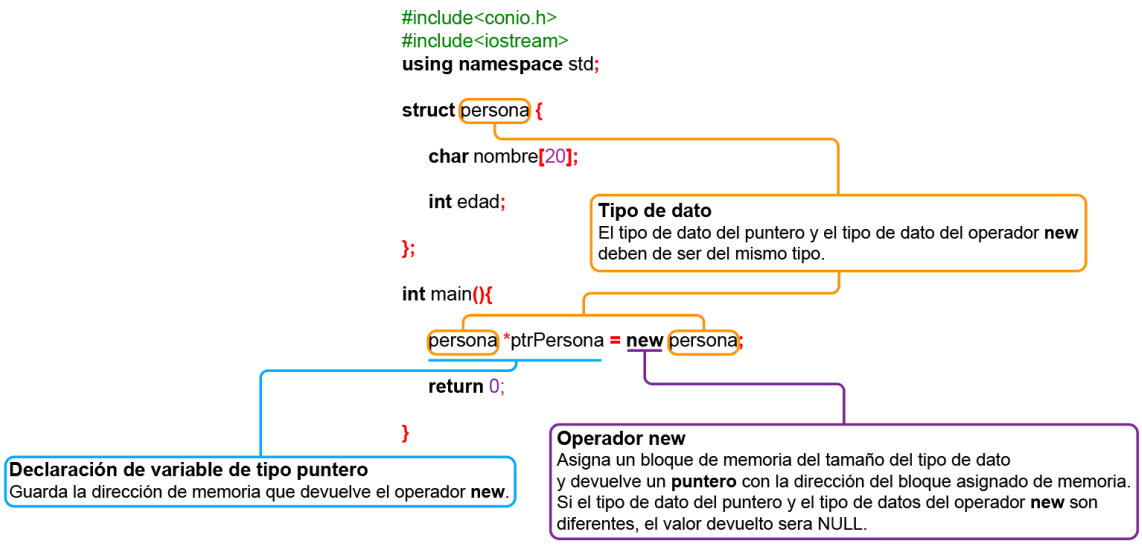


Figura 5. Plan de programación, Operador new

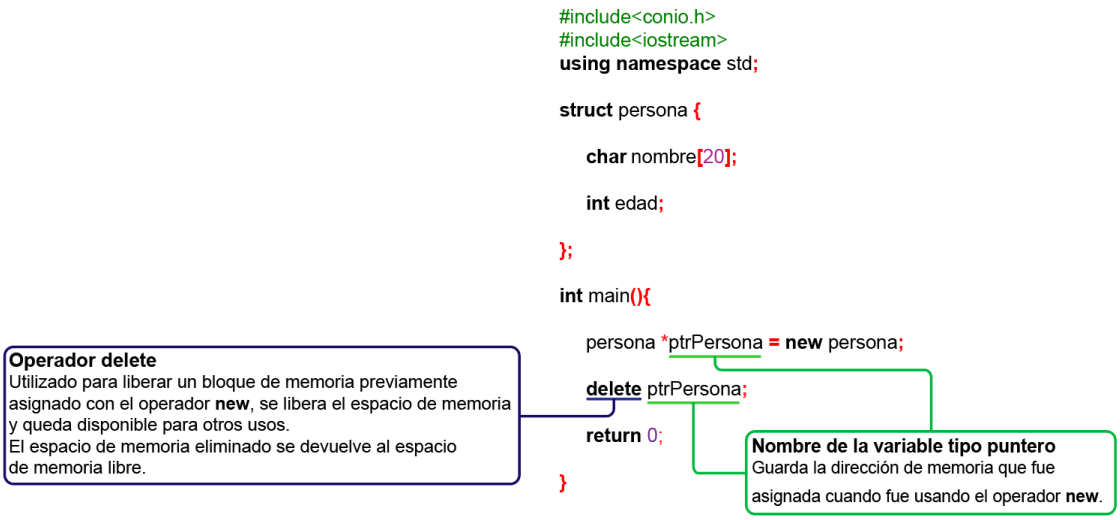


Figura 6. Plan de programación, Operador delete

3.1.1.5. Listas ligadas

Las listas ligadas son una colección de estructuras autoreferenciadas, denominadas nodos, cada uno conectado al siguiente por medio de un enlace, un apuntador. Los nodos son almacenados en posiciones de memoria no contiguas, por lo que cada nodo necesita almacenar la dirección de memoria del siguiente elemento de la lista, para ello se hace uso de los apuntadores y las estructuras autoreferenciadas.

Las operaciones básicas que se pueden llevar a cabo sobre las listas ligadas son: insertar, eliminar y buscar un nodo, modificar la información de un nodo de la lista e imprimir la lista ligada.

3.1.1.6. Listas simplemente ligadas

En las listas simplemente ligadas cada nodo contiene un único enlace que está conectado al nodo siguiente.

Ver anexo 3 para visualizar los ejemplos resueltos creados para las listas simplemente ligadas.

3.1.2. Diseño de la herramienta web

El aspecto visual de la herramienta web, se desarrolló siguiendo la teoría de la carga cognitiva (Sweller et al., 1998), especialmente aplicando el efecto de atención dividida (Ayres & Sweller, 2005). Diseñando la herramienta web en cuatro secciones (ver Figura 7), sección de visualización de la lista simplemente ligada, sección de ejecución de código, sección de acciones y sección de planes de programación, con el fin de evitar los problemas descritos en el efecto de atención dividida y causar en el estudiante un aprendizaje óptimo. Se busca que el estudiante integre mentalmente la implementación en lenguaje C con el funcionamiento interno de las listas simplemente ligadas.

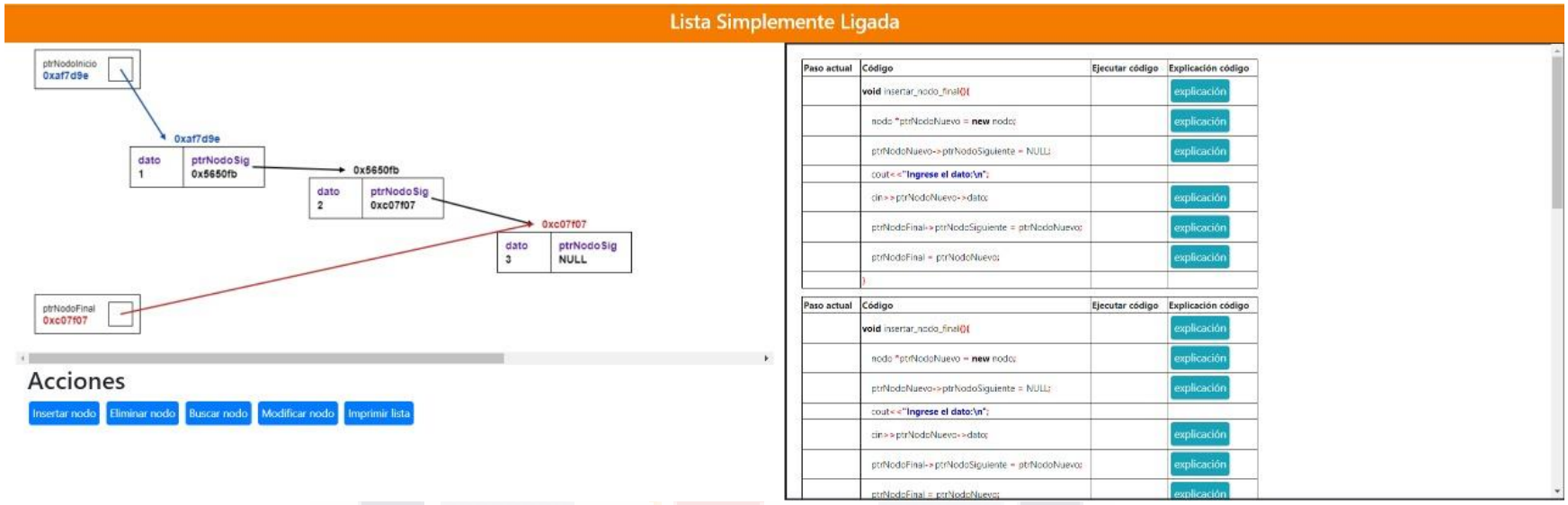


Figura 7 Herramienta web

El diseño de la herramienta web fue hecho de esta manera debido a la estrecha relación que cada sección guarda con las demás, como se explica en seguida, en la sección de acciones el estudiante puede elegir que función desea ejecutar, haciendo que se muestre la función en lenguaje C en la sección de ejecución de código, y así después ir ejecutando línea por línea en la función y visualizando la ejecución del código en la sección de visualización de la lista simplemente ligada. La sección de plan de programación se muestra en el momento de que el alumno quiere conocer lo que hace una línea de código en específico, presionando el botón de explicación.

3.1.2.1. Sección de visualización de lista simplemente ligada

Esta sección fue creada con el objetivo de que el alumno observe, analice y comprenda lo que pasa internamente al momento de ejecutar las funciones de las listas simplemente ligadas en lenguaje C (ver Figura 8).

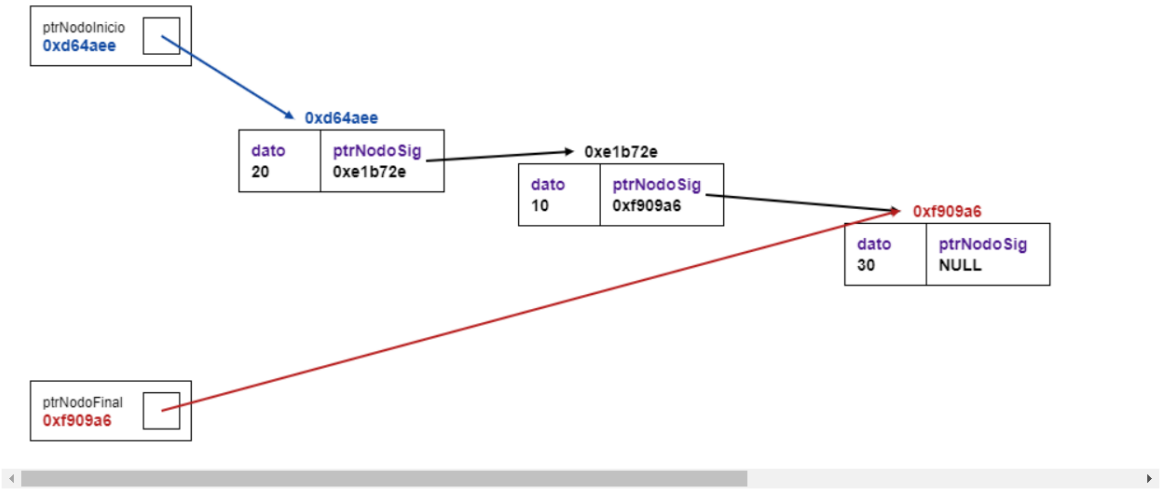


Figura 8. Sección de visualización de listas simplemente ligadas

3.1.2.2. Sección de ejecución de código

En esta sección se muestran las funciones en lenguaje C (ver Figura 9), que el estudiante va seleccionando en la sección de acciones. Aquí el estudiante debe ejecutar línea por línea para observar lo que la línea de código realiza en la sección de visualización de la lista simplemente ligada. Presionando el botón de explicación, el estudiante puede acceder al plan de programación de la línea de código seleccionada.

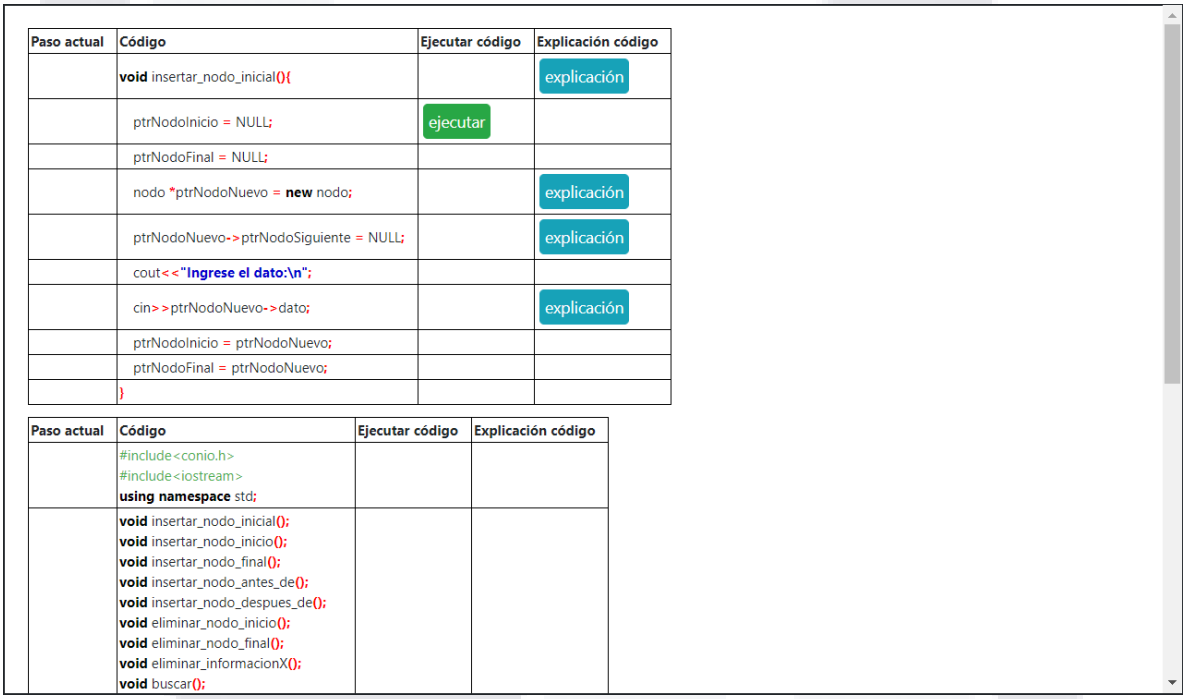


Figura 9. Sección de ejecución de código

3.1.2.3. Sección de acciones

En esta sección se muestran las diferentes funciones que pueden ser implementadas sobre una lista simplemente ligada. Como se había comentado con anterioridad, existen 5 funciones principales: insertar, eliminar, buscar y modificar un nodo e imprimir la lista (ver Figura 10).

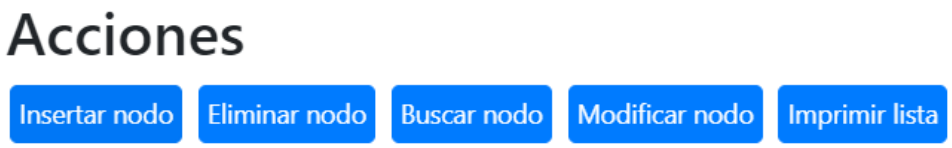


Figura 10. Funciones principales

Para las funciones que integran la función de insertar nodo ver Figura 11 y Figura 12.

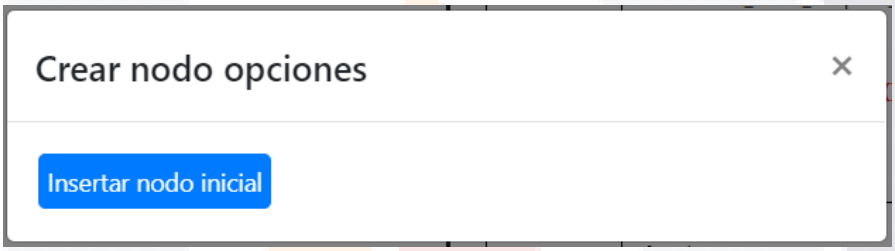


Figura 11. Función insertar nodo inicial

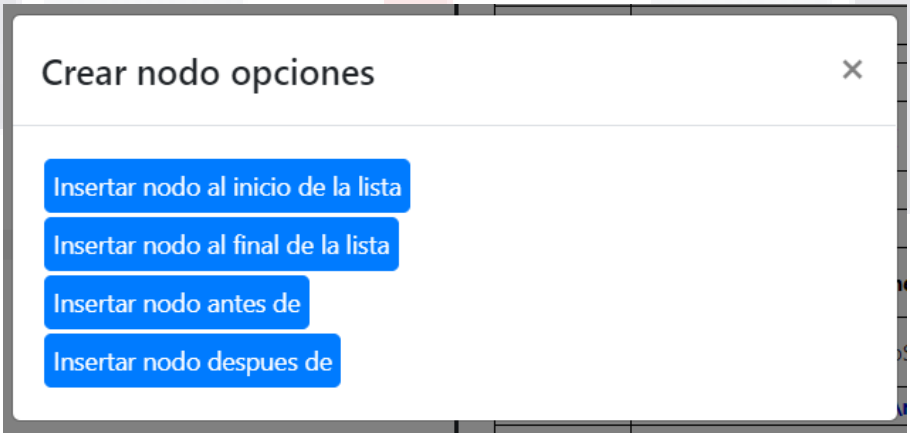


Figura 12. Funciones insertar nodo

Para las funciones que corresponden a eliminar nodo ver Figura 13.

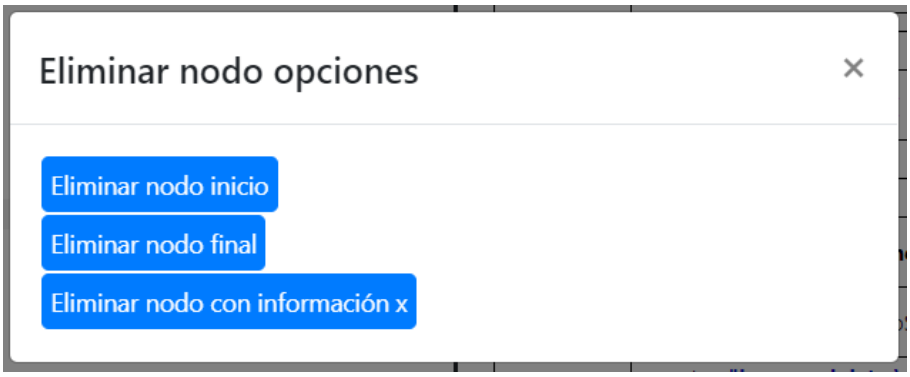


Figura 13. Funciones eliminar nodo

La función para buscar nodo se observa en la Figura 14.

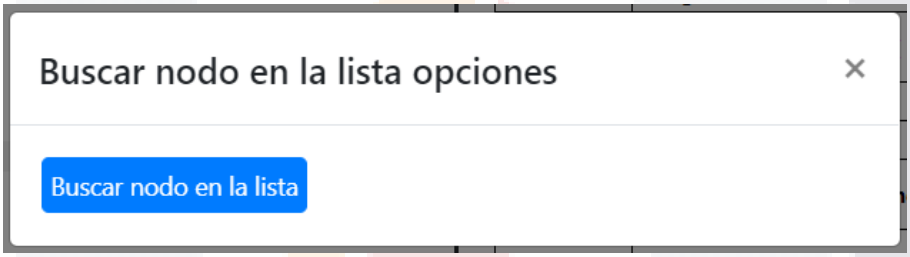


Figura 14. Funciones buscar nodo

Para la función de modificar el valor de un nodo ver Figura 15.

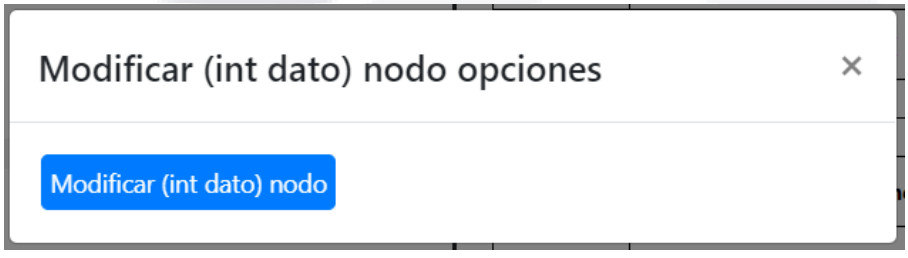


Figura 15. Funciones modificar valor de un nodo

Y finalmente para la funcion de imprimir lista ver Figura 16.

Figura 16. Funciones de imprimir la lista simplemente ligada

3.1.2.4. Sección de plan de programación

Esta sección fue creada para explicar lo que hace cada línea de código en las diversas funciones que se muestran. Para acceder a esta sección basta con que el alumno presione el botón de explicación (ver Figura 17).

Figura 17. Visualizar plan de programación

Se despliega una ventana emergente (ver Figura 18), la cual contiene la explicación de la línea de código seleccionada, así como su plan de programación correspondiente.

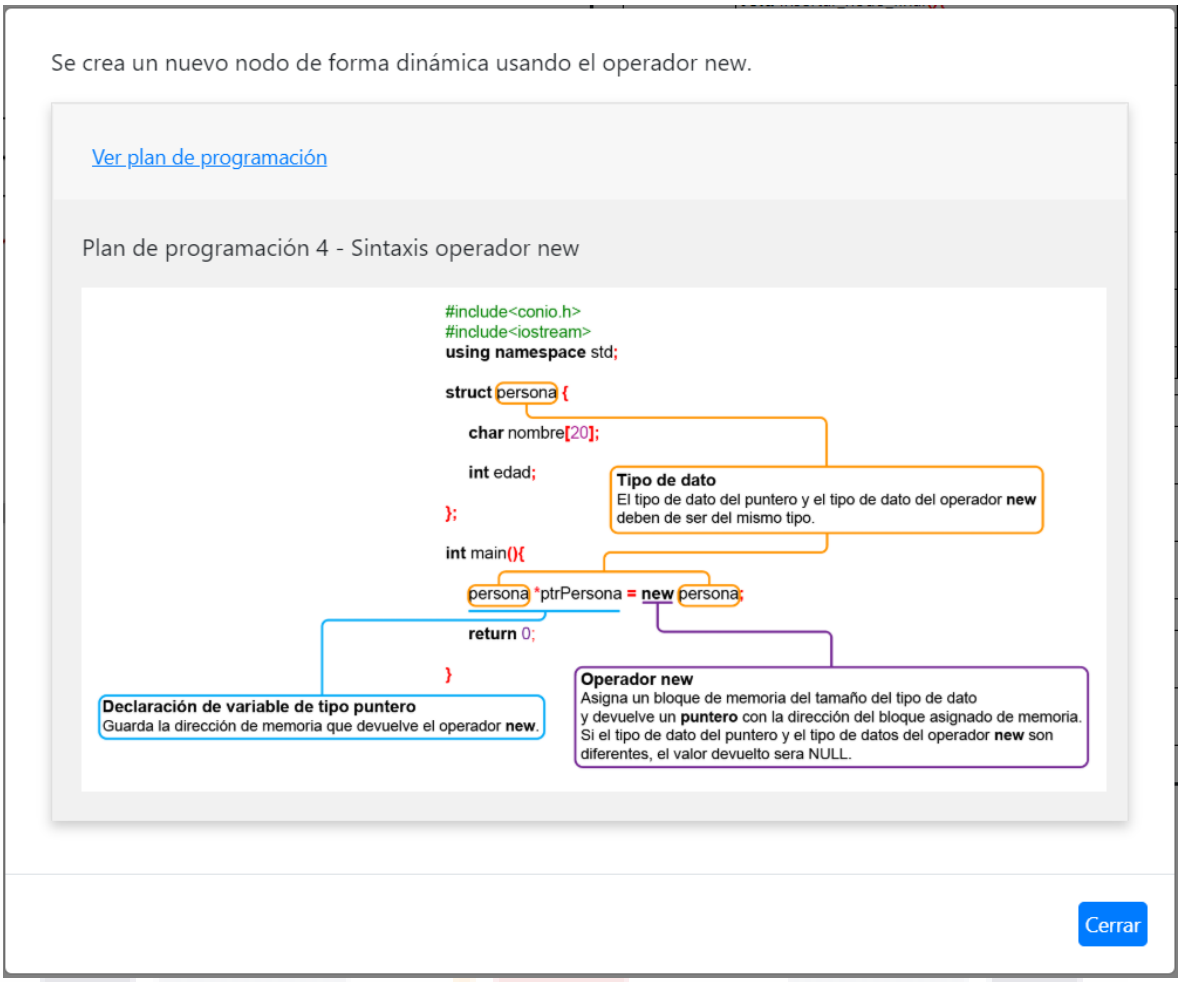


Figura 18. Plan de programación

3.1.3. Desarrollo de la herramienta web

Implementar la funcionalidad de la aplicación fue un gran reto, no sólo era mostrar gráficamente una lista simplemente ligada, con los nodos y sus conexiones, sino que, además, se debía de visualizar lo que se iba ejecutando en la sección de código, con la finalidad de que el alumno entendiera lo que pasa internamente en la computadora cuando ejecuta una pieza de código.

Con el fin de llevar un mejor control sobre la arquitectura del software, se decidió desarrollar la aplicación con el *framework* Laravel versión 5.8 usando PHP versión

7.1.3. Además de que se vuelve más sencillo agregar nuevos módulos en caso de seguir trabajando en un futuro con el desarrollo de la aplicación.

La funcionalidad de los botones, la implementación y visualización del código de la lista simplemente ligada, la visualización los planes de programación y el esquema de la lista simplemente ligada, se programó con JavaScript y JQuery.

La visualización de la lista simplemente ligada se consiguió utilizando la etiqueta Canvas (ver Figura 19) ya que permite dibujar gráficos o realizar animaciones sencillas en combinación con JavaScript.

```
<canvas id="canvas" width="1500" height="400"></canvas>
```

Figura 19. Etiqueta Canvas.

La simulación de la lista simplemente ligada en JavaScript no fue una tarea sencilla, se requirió analizar a profundidad el funcionamiento de las listas simplemente ligadas en lenguaje C y asi transladar la misma funcionalidad a lenguaje JavaScript.

Para el control de la lista simplemente ligada se utilizó una variable tipo “object” en JavaScript (ver Figura 20), la cual funciona como una lista en la que se van agregando nuevos elementos y asociandolos con un identificador unico, en este caso una dirección de memoria ficticia. la Figura 21 muestra la función para generar una dirección de memoria ficticia.

```
objetos = {};
```

Figura 20. Variable Objeto en JavaScript.

```
function generarDireccionMemoria(){

    let hex = "0123456789abcdef";
    let dirMem = "0x";

    for (let i = 0; i < 6; i++){
        dirMem += hex[(Math.floor(Math.random() * 16))];
    }

    return dirMem.trim();

}
```

Figura 21. Función para generar una dirección de memoria ficticia.

En la Figura 22 se señalan los principales atributos que debe de contener el nuevo nodo. Se asigna un identificador único, las variables key_objecto y direccion_memoria contienen el mismo valor. Es importante señalar que se debe de asociar al nuevo elemento con el identificador único (key_objeto) para poder acceder a los atributos de ese objeto. El atributo “ptrLigaSig”, es un objeto, guarda en su atributo “key” la dirección de memoria o en otras palabras el identificador único del elemento. En caso de que sea el ultimo nodo de la lista tendrá el valor de null.

```
objetos[key_objecto] = {

    "key": key_objecto,
    "direccion_memoria": direccion_memoria,

    "dato": {
        "valor": null,
    },

    "ptrLigaSig": {
        "inicializado": false,
        "key": null,
    },
```

Figura 22. Atributos del nodo.

Para guardar la dirección de memoria del nodo inicial y final de la lista se crearon dos variables tipo “object”, (ver Figura 23 y Figura 24) respectivamente, las cuales guardan en su atributo “nodo_conectado” la dirección de memoria (el identificador único) del elemento que esta guardado en la variable objetos.

```
var obj_ptrNodoInicio = {

    "contenedor_principal": {
        "x": 10,
        "y": 10,
        "width": 137,
        "height": 50,
    },

    "disenio": {
        "margen_lineWidth": '2',
        "margen_color": '#000',
        "relleno_color": '#fff',
    },

    "ptrLiga": {
        "x": 107,
        "y": 20,
        "width": 30,
        "height": 30,
    },

    "texto_disenio": {
        "font": '12px Arial',
        "color": '#000',
        "size": 12,
        "espacio": 2,
        "font_valor": 'bold 14px Arial',
        "direccion_memoria_color": '#0d47a1',
    },

    "nombre": "ptrNodoInicio",

    "declarada": false,

    "nodo_conectado": {
        "key": null,
    },

};
```

Figura 23. Variable "object" obj_ptrNodoInicio.

```
var obj_ptrNodoFinal = {

    "contenedor_principal": {
        "x": 10,
        "y": 331,
        "width": 137,
        "height": 50,
    },

    "disenio": {
        "margen_lineWidth": '2',
        "margen_color": '#000',
        "relleno_color": '#fff',
    },

    "ptrLiga": {
        "x": 107,
        "y": 341,
        "width": 30,
        "height": 30,
    },

    "texto_disenio": {
        "font": '12px Arial',
        "color": '#000',
        "size": 12,
        "espacio": 2,
        "font_valor": 'bold 14px Arial',
        "direccion_memoria_color": '#b71c1c',
    },

    "nombre": "ptrNodoFinal",

    "declarada": false,

    "nodo_conectado": {
        "key": null,
    },

};
```

Figura 24. Variable "object" obj_ptrNodoFinal.

3.1.4. Diseño experimental

Para el diseño del experimento y la recolección de datos, se optó por un diseño cuasiexperimental, no aleatorizado, con grupos predefinidos, para facilitar la logística de recolección de datos y la alineación con los temas impartidos durante la materia del plan de estudios de estructura de datos, utilizando dos grupos previamente conformados de 3º semestre de Ingeniería en Sistemas computacionales (ISC). Se seleccionó de manera aleatoria al grupo C, como grupo de experimental (n=36) y el 3º A (n=35) como grupo de control. Para el grupo

experimental, se impartió una sesión de laboratorio de una hora de duración para explicar el uso de la herramienta. A ambos grupos se les proporcionaron 10 ejercicios prediseñados para el experimento sobre escritura de código de **listas simplemente ligadas**, distribuidos en 3 niveles de dificultad (ver anexo 2).

El tema de listas simplemente ligadas se imparte en la 4ª Unidad del curso de estructuras de datos del plan de estudios de ISC. Estos ejercicios fueron revisados por la coordinación de la academia de programación de la carrera de ISC. Se otorgó una semana de práctica (3ª semana de octubre de 2019) para resolver los ejercicios a manera de tarea a ambos grupos. Al grupo experimental se le solicitó resolver los ejercicios utilizando la herramienta de visualización diseñada. El grupo de control resolvió los ejercicios de forma tradicional (estudiando código de ejemplo previamente explicado y proporcionado por el instructor).

Para la recolección de datos se utilizó un diseño Pre prueba – Post prueba, de acuerdo con la Figura 25. Los instrumentos de medición fueron exámenes diseñados por la academia de programación de la carrera de ISC. La primera medición (Pre), correspondió al 2do examen parcial formal curricular y la segunda (Post) fue un examen, también diseñado por la academia, con reactivos solamente del tema de listas simplemente ligadas.

Diseño experimental			
G_1	O_1 (2do Parcial)	--	O_3 (examen de listas ligadas)
G_2	O_2 (2do Parcial)	Tratamiento Experimental	O_4 (examen de listas ligadas)

Figura 25. Esquema de diseño experimental aplicado.

Se definió como variable dependiente el "**aprendizaje e implementación en código del concepto de listas simplemente ligadas**". La variable independiente se definió como el "**método de aprendizaje**" del tema de listas simplemente ligadas, con dos valores: tradicional y experimental.

La variable "**estilo de enseñanza**" se controló para ambos grupos, en el sentido de que se buscaron dos grupos con el mismo instructor. De igual forma, el material didáctico utilizado (ejemplos, ejercicios, presentaciones) fue el mismo para ambos grupos. La variable asociada al conocimiento previo no pudo ser controlada, ya que depende del promedio de calificación obtenido en cursos previos relacionados con programación.

CAPÍTULO IV

RESULTADOS

Los resultados de las pruebas realizadas con los grupos experimental y de control, para verificar el efecto de la utilización de la herramienta web desarrollada utilizando los principios de diseño del efecto de atención dividida y ejemplos resueltos, de la teoría de carga cognitiva, y bajo las condiciones experimentales indicadas en la sección anterior, se muestran a continuación.

En la Tabla 6 se muestran los resultados obtenidos por el grupo de control (n=35) en las pruebas Pre y Post de conocimiento del tema de Listas Simplemente Ligadas, quienes utilizaron métodos de estudio tradicionales (ejemplos, ejercicios y exposiciones del material didáctico del profesor). La media del examen Pre fue de 8.68 y la del examen Post fue de 9.57, en donde se observó un incremento de 0.88 puntos en una escala de 1 a 10, lo que se considera marginal. Cabe resaltar que la moda tanto del examen Pre, como del Post fue de 10.

Al realizar una inspección visual de las distribuciones de frecuencias del grupo de control puede observarse una distribución aproximadamente normal y semejante en los exámenes Pre-Post, lo que sugiere uniformidad entre una y otra medición.

Tabla 6. Estadística descriptiva, resultados Pre-Post tratamiento grupo de control.

Estadísticos

		GpoCtrol_Pre	GpoCtrol_Post
N	Válidos	35	35
	Perdidos	1	1
Media		8.6857	9.5714
Mediana		9.0000	10.0000
Moda		10.00	10.00
Desv. típ.		1.74510	.77784
Varianza		3.045	.605
Asimetría		-1.880	-1.842
Error típ. de asimetría		.398	.398
Curtosis		3.446	2.803
Error típ. de curtosis		.778	.778

Para corroborar lo anterior, se realizó una prueba t de muestras dependientes (ver Tabla 7), es decir, la comparación de medias de cada participante entre la medición Pre y la medición Post (ver Figura 26). El resultado de la comparación de medias (p=0.015) señala que ambas muestras no son estadísticamente diferentes.

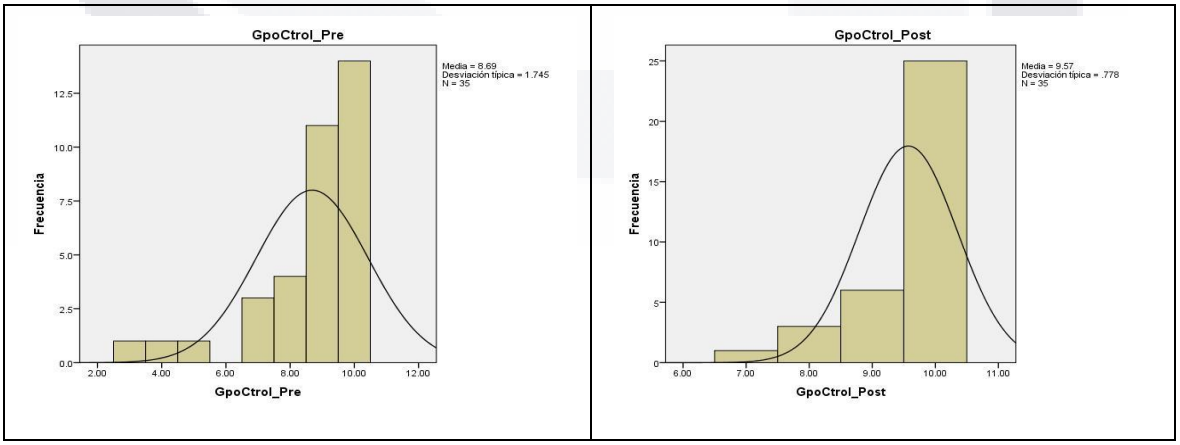


Figura 26. Histogramas de frecuencias, resultados Pre-Post tratamiento, grupo de control

Tabla 7. Prueba t, diferencia de medias, grupo de control, Pre-Post tratamiento.

Prueba de muestras relacionadas

		Diferencias relacionadas					t	gl	Sig. (bilateral)
		Media	Desviación típ.	Error típ. de la media	95% Intervalo de confianza para la diferencia Inferior	Superior			
Par 1	GpoCtrol_Pre - GpoCtrol_Post	-.88571	2.04035	.34488	-1.58660	-.18483	-2.568	34	.015

El grupo experimental (n=36), que como se mencionó utilizó la herramienta Web para aprendizaje de listas simplemente ligadas, basada en el efecto de atención dividida, obtuvo los resultados detallados en la Tabla 8, en donde la media del examen Pre fue de 5.58 y la del examen Post fue de 8.63. Entre ambas mediciones se observó un incremento de 3.05 puntos en una escala de 1 a 10, que se considera significativa. Se resalta que la moda del examen Pre fue de 6 y la del examen Post fue de 10.

Tabla 8. Estadística Descriptiva, resultados Pre-Post tratamiento grupo experimental.

Estadísticos

		GpoExp_Pre	GpoExp_Post
N	Válidos	36	36
	Perdidos	0	0
Media		5.5833	8.6389
Mediana		6.0000	9.5000
Moda		6.00	10.00
Desv. típ.		1.20416	1.85400
Varianza		1.450	3.437
Asimetría		.157	-1.229
Error típ. de asimetría		.393	.393
Curtosis		-.079	.182
Error típ. de curtosis		.768	.768

La inspección visual de las distribuciones de frecuencias (ver Figura 27) muestra una curtosis aproximadamente “leptocúrtica” en los resultados Pre del grupo

experimental y una “platicúrtica” en los resultados Post, intuyéndose una diferencia estadística entre ambas mediciones.

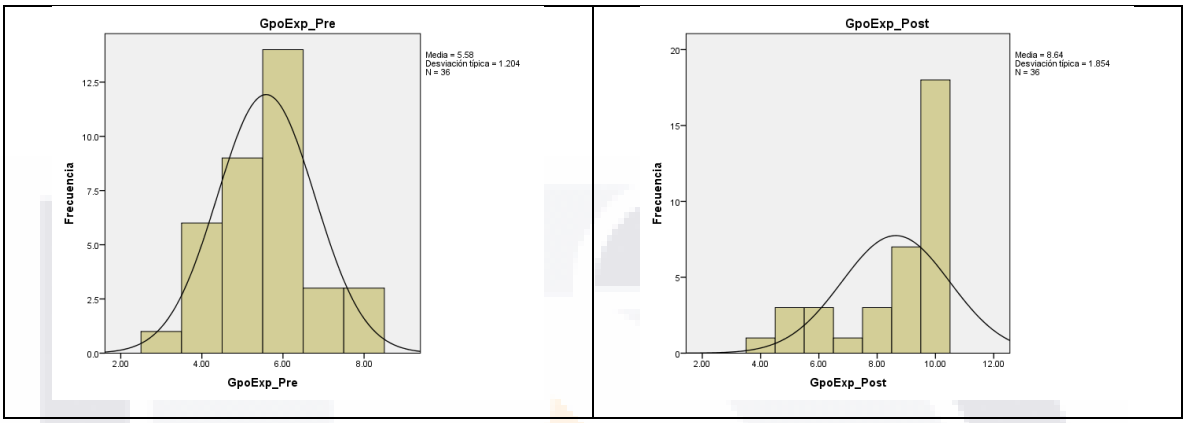


Figura 27. Histograma de frecuencias, resultados Pre-Post tratamiento, grupo experimental

Para corroborar lo anterior, se aplicó nuevamente una prueba t de diferencia de medias (ver Tabla 9) entre las mediciones Pre y Post del grupo experimental, obteniéndose un valor significativo (p=.000) entre ambos exámenes, lo cual a su vez sugiere que el tratamiento aplicado al grupo experimental tuvo un efecto positivo.

Tabla 9. Prueba t de diferencia de medias, muestras relacionadas, Pre-Post tratamiento, grupo experimental

Prueba de muestras relacionadas

		Diferencias relacionadas					t	gl	Sig. (bilateral)
		Media	Desviación típ.	Error típ. de la media	95% Intervalo de confianza para la diferencia				
					Inferior	Superior			
Par 1	GpoExp_Pre - GpoExp_Post	-3.05556	2.09686	.34948	-3.76503	-2.34608	-8.743	35	.000

Finalmente, un análisis de diferencia de medias mediante una prueba t, para comparar los resultados de calificaciones “Post” entre los grupos Experimental y de

control, indica que hay una diferencia estadísticamente significativa entre ambos (p=0.00) (ver Tabla 10).

Tabla 10. Prueba t de diferencia de medias para muestras independientes, grupos control y experimental

Prueba de muestras independientes

		Prueba de Levene para la igualdad de varianzas		Prueba T para la igualdad de medias						
									95% Intervalo de confianza para la diferencia	
		F	Sig.	t	gl	Sig. (bilateral)	Diferencia de medias	Error típ. de la diferencia	Inferior	Superior
Cal	Se han asumido varianzas iguales	20.785	.000	2.749	69	.008	.93254	.33919	.25588	1.60920
	No se han asumido varianzas iguales			2.777	47.227	.008	.93254	.33581	.25707	1.60801

Las medias de ambos grupos, en los resultados “post” difieren en aproximadamente 1 punto, en la escala del 1 al 10, en donde tal diferencia puede atribuirse a diferencias educativas de rendimiento académico y de conocimientos previos de programación, en donde el grupo de control tuvo un mejor rendimiento que el grupo experimental (ver Figura 28). Sin embargo, puede argumentarse que el rendimiento académico del grupo experimental, al inicio del experimento, también resultaba significativamente menor siendo la media de la medición “Pre”=5.58 y la media la medición “Post”=8.64 y la brecha académica entre un grupo de buen rendimiento académico (Ctrol) se redujo a un solo punto de la escala de calificación.

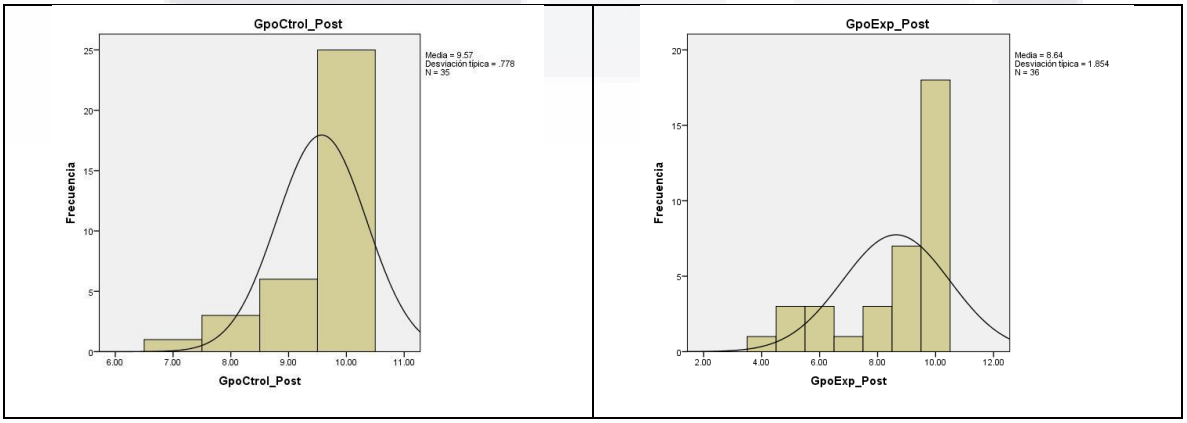


Figura 28. Histograma de calificaciones obtenidas en prueba 'Post' en grupos de control y experimental

CONCLUSIONES

Conclusiones de los resultados del estudio

La aplicación de la prueba t entre el examen Pre y Post del grupo experimental, arrojó un valor significativo (p=.000), indicando que el tratamiento aplicado al grupo experimental tuvo un efecto positivo.

Se puede afirmar que el uso de la herramienta web diseñada a partir de la teoría de la carga cognitiva, y el uso de ejercicios basados en los planes de programación y en el efecto de problemas por completar tuvo un impacto positivo en el aprendizaje de los estudiantes.

Basado en los resultados positivos mostrados en esta investigación y en trabajos anteriores, se concluye que la creación de material instruccional basado en la teoría de la carga cognitiva tiene un efecto positivo en el aprendizaje, en este caso, de listas simplemente ligadas. Se espera que, para futuras continuaciones de este estudio, se seleccionen y se agreguen a la herramienta web otras estructuras de datos, lo que permita ampliar el catálogo de temas para formar una herramienta más robusta, que sirva tanto a estudiantes como a alumnos como material de apoyo para el aprendizaje.

Conclusiones de los objetivos del estudio

El objetivo general definido para este trabajo de investigación fue “diseñar e implementar material didáctico basado en la teoría de la carga cognitiva, particularmente usando el ***efecto del problema por completar***, en combinación con la identificación de ***planes de programación***, para la materia de estructura de datos con el objetivo de medir la efectividad de dicho material en el aprendizaje de

los alumnos que estudian materias afines a las ciencias computacionales de la Universidad Autónoma de Aguascalientes", objetivo que se alcanzó de manera exitosa, logrando resultados positivos.

Se cumplió cada objetivo planteado al comienzo de esta investigación, se logró la correcta identificación y creación de los planes de programación que son esenciales para implementar las Listas Simplemente Ligadas, especificados en el Capítulo II, en dicho capitulo se explican detalladamente cada uno de los planes de programación identificados, los cuales fueron la pieza clave para el desarrollo de la herramienta de apoyo a la enseñanza, desarrollada a partir de la Teoría de la Carga Cognitiva.

Respondiendo las preguntas de investigación, se concluye que, sin importar la complejidad del tema de programación a enseñar, se pueden crear herramientas basadas en nuevas estrategias didácticas, como lo es la Teoría de la Carga Cognitiva, que faciliten la adquisición de conocimiento, ayudando a los estudiantes a generar modelos mentales para tener una mejor comprensión de los algoritmos enseñados.

Cabe resaltar que la herramienta desarrollada como material de apoyo para la materia de Estructuras de Datos, específicamente para el tema de Listas Simplemente Ligadas, arrojo resultados positivos en los estudiantes, al mejorar la compresión y adquisición de conocimiento sobre dicho tema, viéndose reflejados los resultados en las calificaciones obtenidas en el examen que se realizado después de hacer uso de la herramienta.

Por lo que se concluye que es posible crear y desarrollar herramientas didácticas para el aprendizaje basadas en la Teoría de la Carga Cognitivo, las cuales ayudan a mejorar el rendimiento académico de los estudiantes, facilitando la comprensión y adquisición de nuevo conocimiento.

BIBLIOGRAFIA

Adelson, B. (1981). Problem solving and the development of abstract categories in programming languages. *Memory & Cognition*, *9*(4), 422–433. https://doi.org/10.3758/BF03197568

Aguilar, L. J. (2006). *PROGRAMACION EN C++: UN ENFOQUE PRACTICO*. McGraw-Hill Interamericana de España S.L.

Almeida, F., Blanco, V., & Moreno, L. (2004). *EDApplets: Una Herramienta Web para la Enseñanza de Estructuras de datos y Técnicas Algorítmitcas*.

Alty, J. L. (2002, febrero 22). Dual Coding Theory and Computer Education: Some Media Experiments To Examine the Effects of Different Media on Learning. *ED-MEDIA 2002*.

Andersen, S., Mikkelsen, P., Konge, L., Cayé-Thomasen, P., & Sørensen, M. (2016). The effect of implementing cognitive load theory-based design principles in virtual reality simulation training of surgical skills: A randomized controlled trial. *Advances in Simulation*, *1*. https://doi.org/10.1186/s41077-016-0022-1

Ayres, P., & Sweller, J. (2005). The Split-Attention Principle in Multimedia Learning. En *The Cambridge handbook of multimedia learning.* (pp. 135–146). Cambridge University Press. https://doi.org/10.1017/CBO9780511816819.009

Bonar, J., & Soloway, E. (1985). Preprogramming Knowledge: A Major Source of Misconceptions in Novice Programmers. *Human-Computer Interaction*, *1*, 133–161. https://doi.org/10.1207/s15327051hci0102_3

Caspersen, M. E., & Bennedsen, J. (2007). Instructional Design of a Programming Course: A Learning Theoretic Approach. *Proceedings of the Third International Workshop on*

Computing Education Research, 111–122. https://doi.org/10.1145/1288580.1288595

Chang, K., Chiao, B.-C., Chen, S.-W., & Hsiao, R.-S. (2000). A programming learning system for beginners—A completion strategy approach. *Education, IEEE Transactions on*, *43*, 211–220. https://doi.org/10.1109/13.848075

Chase, W. G., & Simon, H. A. (1973). Perception in chess. *Cognitive Psychology*, *4*(1), 55–81. https://doi.org/10.1016/0010-0285(73)90004-2

Connolly, C., Murphy, E., & Moore, S. (2009). Programming Anxiety Amongst Computing Students #x2014;A Key in the Retention Debate? *IEEE Transactions on Education*, *52*(1), 52–56. https://doi.org/10.1109/TE.2008.917193

Cooper, G., & Sweller, J. (1987). Effects of schema acquisition and rule automation on mathematical problem-solving transfer. *Journal of Educational Psychology*, *79*(4), 347–362. https://doi.org/10.1037/0022-0663.79.4.347

Deitel, H. M., & Deitel, P. J. (2004). *Como programar en C/C++y Java*. Pearson Educación.

Du Boulay, B. (1986). Some Difficulties of Learning to Program. *Journal of Educational Computing Research*, *2*(1), 57–73. https://doi.org/10.2190/3LFX-9RRF-67T8-UVK9

Egan, D. E., & Schwartz, B. J. (1979). Chunking in recall of symbolic drawings. En *Memory & Cognition* (Vol. 7, Número 2, pp. 149–158). Psychonomic Society. https://doi.org/10.3758/BF03197595

Gerjets, P., Scheiter, K., & Cierniak, G. (2009). The Scientific Value of Cognitive Load Theory: A Research Agenda Based on the Structuralist View of Theories. *Educational Psychology Review*, *21*(1), 43–54. https://doi.org/10.1007/s10648-008-9096-1

Herrera Loyo, A. (2018). Effects on the School Performance of Teaching Programming in Elementary and Secondary Schools. En S. N. Pozdniakov & V. Dagienė (Eds.),

Informatics in Schools. Fundamentals of Computer Science and Software Engineering (pp. 30–41). Springer International Publishing.

Hromkovič, J., & Lacher, R. (2017). The Computer Science Way of Thinking in Human History and Consequences for the Design of Computer Science Curricula. En V. Dagienė & A. Hellas (Eds.), *Informatics in Schools: Focus on Learning Programming* (pp. 3–11). Springer International Publishing.

Kalyuga, S. (2011). Cognitive Load Theory: How Many Types of Load Does It Really Need? *Educational Psychology Review*, *23*(1), 1–19. https://doi.org/10.1007/s10648-010-9150-7

Kirschner, P. A., Sweller, J., & Clark, R. E. (2006). Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. En *Educational Psychologist*.

Lawrence, R. (2004). *Teaching Data Structures Using Competitive Games* (Vol. 47). https://doi.org/10.1109/TE.2004.825053

Leppink, J., Paas, F., Gog, T., Van der Vleuten, C., & Van Merrienboer, J. J. G. (2014). Effects of pairs of problems and examples on task performance and different types of cognitive load. *Learning and Instruction*, *30*, 32–42. https://doi.org/10.1016/j.learninstruc.2013.12.001

Matthíasdóttir, Á. (2006). *How to teach programming languages to novice students? Lecturing or not?*

New South Wales. Centre for Education Statistics and Evaluation. (2017). *Cognitive Load Theory: Research that Teachers Really Need to Understand*. New South Wales. Centre for Education Statistics and Evaluation.

Niesser, U. (1976). *Cognition and Reality: Principles and Implications of Cognitive Psychology*. W.H. Freeman.

Osvaldo Cairó, & Silvia Guardati. (2006). *Estructuras de datos* (3a.). McGraw-Hill.

Paas, F., Renkl, A., & Sweller, J. (2004). Cognitive Load Theory: Instructional Implications of the Interaction between Information Structures and Cognitive Architecture. *Instructional Science*, *32*(1), 1–8. https://doi.org/10.1023/B:TRUC.0000021806.17516.d0

Paas, F., van Gog, T., & Sweller, J. (2010). Cognitive Load Theory: New Conceptualizations, Specifications, and Integrated Research Perspectives. *Educational Psychology Review*, *22*(2), 115–121. https://doi.org/10.1007/s10648-010-9133-8

Rahimi, E., Barendsen, E., & Henze, I. (2017). Identifying Students' Misconceptions on Basic Algorithmic Concepts Through Flowchart Analysis. En V. Dagienė & A. Hellas (Eds.), *Informatics in Schools: Focus on Learning Programming* (pp. 155–168). Springer International Publishing.

Rich, C. (2004). *Inspection Methods in Programming*.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education*, *13*, 137. https://doi.org/10.1076/csed.13.2.137.14200

Rosminah, S., md derus, siti rosminah, Mohamad Ali, A. Z., & Ali, M. (2012). *Difficulties in learning programming: Views of students*. https://doi.org/10.13140/2.1.1055.7441

S. Garner. (2002). The learning of plans in programming: A program completion approach. *International Conference on Computers in Education, 2002. Proceedings.*, 1053–1057 vol.2. https://doi.org/10.1109/CIE.2002.1186149

Schnotz, W., & Kürschner, C. (2007). A Reconsideration of Cognitive Load Theory. *Educational Psychology Review*, *19*(4), 469–508. https://doi.org/10.1007/s10648-007-9053-4

Segura Díaz, C. M., & Pita Andreu, M. I. (2007). *Una herramienta para el Estudio de Estructuras de Datos y Algoritmos*.

Solitro, U., Pasini, M., De Gradi, D., & Brondino, M. (2017). A Preliminary Investigation on Computational Abilities in Secondary School. En V. Dagienė & A. Hellas (Eds.), *Informatics in Schools: Focus on Learning Programming* (pp. 169–179). Springer International Publishing.

Soloway, E., & Ehrlich, K. (1984). Empirical Studies of Programming Knowledge. *Software Engineering, IEEE Transactions on*, *SE-10*, 595–609. https://doi.org/10.1109/TSE.1984.5010283

Sweller, J. (1994). Cognitive load theory, learning difficulty, and instructional design. *Learning and Instruction*, *4*(4), 295–312. https://doi.org/10.1016/0959-4752(94)90003-5

Sweller, J. (2010). Element Interactivity and Intrinsic, Extraneous, and Germane Cognitive Load. *Educational Psychology Review*, *22*(2), 123–138. https://doi.org/10.1007/s10648-010-9128-5

Sweller, J., Ayres, P., & Kalyuga, S. (2011). The Split-Attention Effect. En J. Sweller, P. Ayres, & S. Kalyuga (Eds.), *Cognitive Load Theory* (pp. 111–128). Springer New York. https://doi.org/10.1007/978-1-4419-8126-4_9

Sweller, J., & Cooper, G. A. (1985). The use of worked examples as a substitute for problem solving in learning algebra. *Cognition and Instruction*, *2*(1), 59–89. https://doi.org/10.1207/s1532690xci0201_3

Sweller, J., van Merriënboer, J. J. G., & Paas, F. (2019). Cognitive Architecture and Instructional Design: 20 Years Later. *Educational Psychology Review*, *31*(2), 261–292. https://doi.org/10.1007/s10648-019-09465-5

Sweller, J., van Merrienboer, J. J. G., & Paas, F. G. W. C. (1998). Cognitive Architecture and Instructional Design. *Educational Psychology Review*, *10*(3), 251–296. https://doi.org/10.1023/A:1022193728205

Tan, P. H., Yee, T., & Ling, siew-woei. (2009). *Learning Difficulties in Programming Courses: Undergraduates' Perspective and Perception* (p. 46). https://doi.org/10.1109/ICCTD.2009.188

Tarmizi, R. A., & Sweller, J. (1988). Guidance during mathematical problem solving. *Journal of Educational Psychology*, *80*(4), 424–436. https://doi.org/10.1037/0022-0663.80.4.424

Universidad Autónoma de Aguascalientes. (2019). *Índices de reprobación por materia y deserción semestral.* http://dei.dgpd.uaa.mx/estudios/index.php

van Gog, T., Paas, F., & Sweller, J. (2010). Cognitive Load Theory: Advances in Research on Worked Examples, Animations, and Cognitive Load Measurement. *Educational Psychology Review*, *22*(4), 375–378. https://doi.org/10.1007/s10648-010-9145-4

Van Merriënboer, J. J. G. (1990). Strategies for Programming Instruction in High School: Program Completion vs. Program Generation. *Journal of Educational Computing Research*, *6*(3), 265–285. https://doi.org/10.2190/4NK5-17L7-TWQV-1EHL

van Merriënboer, J. J. G., & Sweller, J. (2005). Cognitive Load Theory and Complex Learning: Recent Developments and Future Directions. *Educational Psychology Review*, *17*(2), 147–177. https://doi.org/10.1007/s10648-005-3951-0

Wang, Z. (2012). The Research on Teaching Ideas of "Data Structure and Algorithm" in Non-computer Major. En A. Xie & X. Huang (Eds.), *Advances in Computer Science and Education* (pp. 249–254). Springer Berlin Heidelberg.

ANEXOS

Anexo 1. Programa de materia de estructuras de datos

UNIVERSIDAD AUTÓNOMA DE AGUASCALIENTES

PROGRAMA DE MATERIA

DATOS DE IDENTIFICACIÓN

MATERIA:	**ESTRUCTURA DE DATOS**				
CENTRO ACADÉMICO:	CIENCIAS BASICAS				
DEPARTAMENTO ACADÉMICO:	CIENCIAS DE LA COMPUTACIÓN				
PROGRAMA EDUCATIVO:	LIC. EN INFORMATICA TECNOLOGIAS COMPUTACIONALES				
AÑO DEL PLAN DE ESTUDIOS:	2014	**SEMESTRE:**	3º	**CLAVE DE LA MATERIA:**	22248
ÁREA ACADÉMICA:	SOFTWARE		**PERIODO EN QUE SE IMPARTE:**	AGO-DIC.	
HORAS SEMANA T/P:	2/3		**CRÉDITOS:**	7	
MODALIDAD EDUCATIVA EN LA QUE SE IMPARTE:	PRESENCIAL		**NATURALEZA DE LA MATERIA:**	TEÓRICO-PRÁCTICA	
ELABORADO POR:	Dra. Elva Díaz Díaz y Dra. Eunice Ponce de León Sentí				
REVISADO Y APROBADO POR LA ACADEMIA DE:	APD, MAML, JCPG		**FECHA DE ACTUALIZACIÓN:**	JUNIO 2016	

DESCRIPCIÓN GENERAL

Se trata de un curso teórico / practica sobre el manejo de las diferentes estructuras de almacenamiento de datos y técnicas de ordenación y búsqueda. El curso es de cinco sesiones a la semana con una duración de una hora cada sesión. La parte teórica representa una aportación frente el aula para avance y aclaración de dudas por parte del maestro, teniendo como necesario que el alumno le dedique mínimo media hora diaria extra clase para la elaboración de tareas, investigación y documentación.

OBJETIVO (S) GENERAL (ES)

CONTENIDOS DE APRENDIZAJE

UNIDAD TEMÁTICA I: INTRODUCCION A LA ESTRUCTURA DE DATOS (8 hrs.)		
OBJETIVOS PARTICULARES	**CONTENIDOS**	**FUENTES DE CONSULTA**
Conocer el significado y términos de información y de Estructura de Datos. Conocer y aplicar las estructuras de arreglos, conjuntos y registros en un lenguaje de programación	1. La información y su significado 1.1 Definición de bit y byte 2. Definición y explicación de almacenamiento de información. 3. Definir estructura de datos. 4. Arreglos 4.1 Definición de Arreglo 4.2 Operaciones con arreglos 4.3 Acceso a los elementos de un vector, matriz, n-dimensiones 5. Registros 5.1 Definición de Registros 5.2 Operaciones con registros 5.3 Acceso a los elementos de un registro	1,2,3
	Diferencias con los arreglos	

*En caso de no aplicar algún elemento, escribir **N/A**

Código: FO-030200-13
Revisión: 02
Emisión: 13/12/11

A

UNIDAD TEMÁTICA II: ESTRUCTURAS DE DATOS SECUENCIALES (8 hrs.)		
OBJETIVOS PARTICULARES	**CONTENIDOS**	**FUENTES DE CONSULTA**
Conocer y aprender el concepto y manejo de los Stacks dentro del ambiente de programación, así como solucionar los problemas de desbordamiento de pilas en los programas. Conocer y aprender el concepto y manejo de colas	1. Definición de Stack (Pilas). 1.1 Representación de una pila y ejemplos 1.2 Operaciones con pilas 1.3 Algoritmos para el manejo de pilas 1.4 Ejemplos del uso de la pila. 1.4.1 Posfix, Prefix, Infix 1.4.2 Algoritmo para la conversión de expresiones infijas a su forma postfija y prefija 2. Definición de cola 2.1 Colas simples 2.1.1 Representación de colas y ejemplos. 2.1.2 Operaciones con colas 2.1.3 Algoritmos para el manejo de colas simples 2.2 Colas circulares 2.2.1 Representación de colas circulares y ejemplos. 2.2.2 Operaciones con colas circulares 2.2.3 Algoritmos para el manejo de colas circulares	1,2,3,4

UNIDAD TEMÁTICA III: RECURSIVIDAD (4 hrs.)		
OBJETIVOS PARTICULARES	**CONTENIDOS**	**FUENTES DE CONSULTA**
prender y practicar el concepto de recursividad	1. Concepto de Recursividad 2. Propiedades de los algoritmos recursivos 3. Funcionamiento interno de la recursividad 4. Uso de pilas para simular la recursividad Ejemplos (torres de Hanoi, multiplicación por sumas, etc.)	1,2,5

UNIDAD TEMÁTICA IV: MEMORIA DINAMICA Y LISTAS ENLAZADAS (8 hrs.)		
OBJETIVOS PARTICULARES	**CONTENIDOS**	**FUENTES DE CONSULTA**
Aprender y manejar los diferentes conceptos de memoria dinámica. Conocer y manejar los diferentes tipos de listas a través de los apuntadores.	1. Definir memoria dinámica. 1.1. Manejo de la memoria dinámica. 1.2. Definición de las operaciones básicas de la memoria dinámica. 1.3. Ejemplos del manejo de memoria dinámica 2. Conceptos de listas 2.1. Concepto de apuntador, nodo y nodo nulo 2.2. Listas Simples 1. Operaciones con listas simples (Algoritmos: insertar al inicio, insertar al final, insertar un elemento intermedio recorrer, borrar al inicio, borrar al final, buscar un elemento) 2.3. Listas Simples Circulares 1. Operaciones con listas simples circulares (Algoritmos: insertar, recorrer, borrar) 2.4. Listas Dobles Operaciones con listas dobles (insertar al inicio, insertar al final, insertar un elemento intermedio recorrer, borrar al inicio, borrar al final, buscar un elemento)	1,2,5

UNIDAD TEMATICA V: ÁRBOLES (8 hrs.)

*En caso de no aplicar algún elemento, escribir **N/A**

Código: FO-030200-13
Revisión: 02
Emisión: 13/12/11

B

OBJETIVOS PARTICULARES	CONTENIDOS	FUENTES DE CONSULTA
1. Aprender y practicar el concepto de árbol y sus aplicaciones, así como realizar el balanceo cuando sea necesario..	1. Introducción a los Árboles 1.1. Definición de árbol. 1.2. Árboles generales 1.2.1. Características y propiedades de los árboles 1.2.2. Longitud de camino interno y externo 2. Tipos de Árboles 2.1. Árboles binarios. 2.1.1. Representación de un árbol general como árbol binario 2.1.2. Inserción de elementos en un árbol binario 2.1.3. Recorridos en un árbol binario (Preorden, Inorden, Postorden) 2.1.4. Borrado de elementos en un árbol binario 2.1.5. Árboles binarios de búsqued 2.2. Árboles Balanceados (AVL) 2.2.1. Definición de un árbol AVL 2.2.2. Inserción de elementos en un árbol balanceado 2.2.3. Factores de equilibrio 2.2.4. Reglas de reestructuración (equilibrio/balanceo) de árboles 2.2.5. Borrado de elementos en un árbol balanceado	1,2,5

UNIDAD TEMÁTICA VI: METODOS DE ORDENACION (12 hrs.)		
OBJETIVOS PARTICULARES	**CONTENIDOS**	**FUENTES DE CONSULTA**
Conocer y entender los conceptos y tipos de ordenamientos de datos	. Tipos de ordenaciones 1.1.Características 1.2.Diferencias 2. Ordenaciones Internas 2.1. Ordenación por Intercambio (burbuja, shaker) 2.2. Ordenación por Quick Sort 3. Ordenaciones Externas 3.1. Intercalación de archivos 3.2. Mezcla de archivos Análisis de eficiencia de los métodos de ordenación	2,5

UNIDAD TEMÁTICA VII: METODOS DE BUSQUEDA (8 hrs.)		
OBJETIVOS PARTICULARES	**CONTENIDOS**	**FUENTES DE CONSULTA**
1. Conocer y aplicar los conceptos de búsquedas.	1. Introducción a las búsquedas. 1.1. Tipos Internos 1.2. Tipos externos Métodos de búsqueda 2.1 Método Secuencial 2.2 Método Secuencial con listas 2.3 Método Binario	1,2,5

METODOLOGÍA DE ENSEÑANZA - APRENDIZAJE

*En caso de no aplicar algún elemento, escribir **N/A**

C

1. Exposiciones verbales por parte del profesor.
2. Realización de trabajos por parte de los alumnos.
3. Investigación de temas y asistencia a congresos.
4. Clases prácticas en el Laboratorio y evaluadas.

RECURSOS DIDÁCTICOS

Presentaciones electrónicas, pintaron, mapas conceptuales y material didáctico.
Uso del Laboratorio de computación.

EVALUACIÓN DE LOS APRENDIZAJES

La evaluación debe ser diagnóstica, formativa y sumaria bajo los siguientes lineamientos:

1. PARTE TEÓRICA: Se realizarán 3 exámenes escritos con la siguiente ponderación:

2 exámenes parciales	20% cada parcial
1 examen final	25%

2. PARTE PRÁCTICA: - Un trabajo final, con una ponderación del 25% (Entregar avances en los periodos de exámenes parciales)
- Prácticas evaluadas en el Laboratorio y tareas, con una ponderación del 10%

NOTA 1: Para tener derecho al examen final es necesario que se entregue el trabajo final.
NOTA 2: Para poder acreditar la materia es necesario aprobar la teoría y la práctica por separado.

FUENTES DECONSULTA

BÁSICAS:

1. - Algoritmos y estructuras de datos
Charles F. Bowman, Sergio Gerardo López Hernández
Año 1999
Oxford University Press ISBN 970-613-459-X

3. - Algoritmos y estructuras de Datos NiklausWirth Año 1987 Pretice Hall ISBN 968-880-113-5

5. -Estructuras de Datos Autores Osvaldo Cairó, Guardati Editorial McGraw-Hill Año 1993 ISBN 970-10-0258-X

2. - Estructura de Datos en C
Aaron M. Tenenbaum, Moshe J. Augenstein
Prentice-Hall Año 1993 ISBN 968-880-256-5

4. - Estructura de datos: Algoritmos, abstracción y objetos Luis Joyanes Aguilar, Ignacio Zahonero Martínez Editorial McGraw Hill Año 2001 ISBN 844-812-042-6

COMPLEMENTARIAS:

Otros Recursos

*En caso de no aplicar algún elemento, escribir **N/A**

D

Anexo 2. Ejercicios de listas simplemente ligadas

Tomando como referencia las funciones de listas simplemente ligadas mostradas en la aplicación http://www.estructuras-de-datos.com/listas-simplemente-ligadas, desarrolle los siguientes ejercicios en código C++, modifique las funciones de ser necesario.

Sección 1

Ejercicio 1

Implementar una función que devuelva la suma de los números introducidos en los nodos de la lista simplemente ligada.

Ejercicio 2

Implementar una función que devuelva el número de nodos en una lista simplemente ligada.

Ejercicio 3

Implementar una función que genere una lista de nombres en forma ascendente

Sección 2

Ejercicio 1

Desarrollar un programa que utilice una **estructura smartphone**, cuyos miembros serán: modelo (string) y precio (double) y utilice un menú de opciones con las operaciones siguientes:

- Crear lista.
- Insertar al final.
- Insertar antes de un elemento de la lista dado como referencia.
- Eliminar un elemento de la lista buscando por su miembro modelo.
- Obtener número de elementos de la lista.
- Salir

Ejercicio 2

Escribir un programa que realice las siguientes tareas:

- Crear una lista enlazada de números enteros positivos al alzar, donde la inserción se realiza por el último nodo.
- Recorrer la lista para mostrar los elementos por pantalla.
- Eliminar todos los nodos que superen un valor dado.

Ejercicio 3

Implementar un procedimiento para insertar un dato (int) en orden ascendente en una lista enlazada. Es decir, el nodo que representa el dato debe ser insertado en una posición tal que al recorrer la lista los nodos se recorran de menor a mayor respecto del dato.

Sección 3

Ejercicio 1

Una librería nos pide que desarrollemos una aplicación que guarde el registro de los **Libros** que tiene en existencia. Cada libro tiene lo siguientes atributos: nombre del libro (string), nombre del autor (string), año de publicación (int) y precio del libro (double).

La aplicación debe de poder guardar un nuevo libro, así como también debe de contener las funciones de eliminar y buscar libros e imprimir la lista completa de libros que estén guardados.

Para la búsqueda de libros se debe de utilizar el atributo nombre del libro.

El formato para la impresión de cada libro puede ser el siguiente:

Nombre del libro: xxxx

Nombre del autor: xxxx

Año de publicación: xxxx

Precio: $xxxx.xx

Ejercicio 2

Una tienda de abarrotes nos pide que desarrollemos una aplicación que registre los **Productos** pasados por el escáner al momento de realizar el cobro de varios productos.

La aplicación debe de almacenar los **Productos**, cada producto al guardarse en la lista debe de contener los atributos: Referencia del producto (int), nombre del producto (string), cantidad del producto (double), precio del producto (double) e Importe(double).

También la aplicación debe de poder modificar la cantidad de los productos guardados previamente buscándolos por su referencia y eliminar productos de igual forma buscándolos por su referencia y debe de mostrar el importe total del ticket (double) al momento de imprimir la lista de productos.

El formato para la impresión de los productos y el importe total puede ser el siguiente:

Referencia **** Cantidad *** Nombre del producto *** Precio *** Importe

Prod1	5.00	producto 1	$10.50	$52.50
Prod2	2.00	producto 2	$29.00	$58

Importe total $110.50

Anexo 3. Ejemplo resuelto de funciones de lista simplemente ligada

```
void insertar_nodo_inicial(){
  ptrNodoInicio = NULL;
  ptrNodoFinal = NULL;
  nodo *ptrNodoNuevo = new nodo;
  ptrNodoNuevo->ptrNodoSiguiente = NULL;
  cout<<"Ingrese el dato:\n";
  cin>>ptrNodoNuevo->dato;
  ptrNodoInicio = ptrNodoNuevo;
  ptrNodoFinal = ptrNodoNuevo;
}
```

```
void insertar_nodo_inicio(){
  nodo *ptrNodoNuevo = new nodo;
  ptrNodoNuevo->ptrNodoSiguiente = NULL;
  cout<<"Ingrese el dato:\n";
  cin>>ptrNodoNuevo->dato;
  ptrNodoNuevo->ptrNodoSiguiente = ptrNodoInicio;
  ptrNodoInicio = ptrNodoNuevo;
}
```

```
void insertar_nodo_final(){
  nodo *ptrNodoNuevo = new nodo;
  ptrNodoNuevo->ptrNodoSiguiente = NULL;
  cout<<"Ingrese el dato:\n";
  cin>>ptrNodoNuevo->dato;
  ptrNodoFinal->ptrNodoSiguiente = ptrNodoNuevo;
  ptrNodoFinal = ptrNodoNuevo;
}
```

```
void insertar_antes_de(){
  nodo *ptrNodoAnterior = NULL;
  nodo *ptrNodoActual = ptrNodoInicio;
  bool encontrado = true;
  int busqueda = 0;
  cout<<"Ingrese el número a buscar:\n";
  cin>>busqueda;
  while(ptrNodoActual->dato != busqueda && encontrado){
    if(ptrNodoActual->ptrNodoSiguiente != NULL){
      ptrNodoAnterior = ptrNodoActual;
      ptrNodoActual = ptrNodoActual->ptrNodoSiguiente;
    }else{
      encontrado = false;
    }
  }
  if(encontrado){
    nodo *ptrNodoNuevo = new nodo;
    ptrNodoNuevo->ptrNodoSiguiente = NULL;
    cout<<"Ingrese el dato:\n";
    cin>>ptrNodoNuevo->dato;
    if (ptrNodoInicio == ptrNodoActual){
      ptrNodoNuevo->ptrNodoSiguiente = ptrNodoInicio;
      ptrNodoInicio = ptrNodoNuevo;
    }else{
      ptrNodoAnterior->ptrNodoSiguiente = ptrNodoNuevo;
      ptrNodoNuevo->ptrNodoSiguiente = ptrNodoActual;
    }
  }else{
    cout<<"El nodo dado como referencia no se encuentra en la lista.\n";
    getch();
  }
}
```

B

```
void insertar_despues_de(){
  nodo *ptrNodoActual = ptrNodoInicio;
  bool encontrado = true;
  int busqueda = 0;
  cout<<"Ingrese el número a buscar:\n";
  cin>>busqueda;
  while(ptrNodoActual->dato != busqueda && encontrado){
    if(ptrNodoActual->ptrNodoSiguiente != NULL){
      ptrNodoActual = ptrNodoActual->ptrNodoSiguiente;
    }else{
      encontrado = false;
    }
  }
  if(encontrado){
    nodo *ptrNodoNuevo = new nodo;
    ptrNodoNuevo->ptrNodoSiguiente = NULL;
    cout<<"Ingrese el dato:\n";
    cin>>ptrNodoNuevo->dato;
    if(ptrNodoFinal == ptrNodoActual){
      ptrNodoFinal->ptrNodoSiguiente = ptrNodoNuevo;
      ptrNodoFinal = ptrNodoNuevo;
    }else{
      ptrNodoNuevo->ptrNodoSiguiente = ptrNodoActual->ptrNodoSiguiente;
      ptrNodoActual->ptrNodoSiguiente = ptrNodoNuevo;
    }
  }else{
    cout<<"El nodo dado como referencia no se encuentra en la lista.\n";
    getch();
  }
}
```

C

```
void eliminar_nodo_inicio(){
  if(ptrNodoInicio != NULL && ptrNodoFinal != NULL){
    nodo *ptrNodoEliminar = ptrNodoInicio;
    if(ptrNodoInicio == ptrNodoFinal){
      ptrNodoInicio = NULL;
      ptrNodoFinal = NULL;
    }else{
      ptrNodoInicio = ptrNodoInicio->ptrNodoSiguiente;
    }
    delete ptrNodoEliminar;
  }else{
    cout<<"La lista esta vacía.\n";
    getch();
  }
}
```

```
void eliminar_nodo_final(){
  if(ptrNodoInicio != NULL && ptrNodoFinal != NULL){
    nodo *ptrNodoEliminar = ptrNodoFinal;
    if(ptrNodoFinal == ptrNodoInicio){
      ptrNodoInicio = NULL;
      ptrNodoFinal = NULL;
    }else{
      nodo *ptrNodoActual = ptrNodoInicio;
      while(ptrNodoActual->ptrNodoSiguiente != ptrNodoFinal){
        ptrNodoActual = ptrNodoActual->ptrNodoSiguiente;
      }
      ptrNodoActual->ptrNodoSiguiente = NULL;
      ptrNodoFinal = ptrNodoActual;
    }
    delete ptrNodoEliminar;
  }else{
    cout<<"La lista esta vacía\n";
    getch();
  }
}
```

D

```
void eliminar_informacionX(){
  if(ptrNodoInicio != NULL && ptrNodoFinal != NULL){
    nodo *ptrNodoActual = ptrNodoInicio;
    nodo *ptrNodoAnterior = NULL;
    bool encontrado = true;
    int busqueda = 0;
    cout<<"Ingrese el (int) dato a buscar:\n";
    cin>>busqueda;
    while(ptrNodoActual->dato != busqueda && encontrado){
      if(ptrNodoActual->ptrNodoSiguiente != NULL){
        ptrNodoAnterior = ptrNodoActual;
        ptrNodoActual = ptrNodoActual->ptrNodoSiguiente;
      }else{
        encontrado = false;
      }
    }
    if(encontrado){
      nodo *ptrNodoEliminar = ptrNodoActual;
      if(ptrNodoEliminar == ptrNodoInicio && ptrNodoEliminar == ptrNodoFinal){
        ptrNodoInicio = NULL;
        ptrNodoInicio = NULL;
      }else{
        if(ptrNodoEliminar == ptrNodoInicio){
          ptrNodoInicio = ptrNodoEliminar->ptrNodoSiguiente;
        }else{
          if(ptrNodoFinal == ptrNodoEliminar){
            ptrNodoFinal = ptrNodoAnterior;
          }
          ptrNodoAnterior->ptrNodoSiguiente = ptrNodoEliminar->ptrNodoSiguiente;
        }
      }
      delete ptrNodoEliminar;
    }else{
      cout<<"El (int) dato "<<busqueda<<" no se encuentra en la lista.\n";
    }
  }else{
    cout<<"La lista esta vacía.\n";
  }
  getch();
}
```

E

```
void buscar(){
  if(ptrNodoInicio != NULL && ptrNodoFinal != NULL){
    nodo *ptrNodoActual = ptrNodoInicio;
    int busqueda = 0;
    cout<<"Ingrese el valor de (int dato) a buscar:\n";
    cin>>busqueda;
    while(ptrNodoActual != NULL && ptrNodoActual->dato != busqueda){
      ptrNodoActual = ptrNodoActual->ptrNodoSiguiente;
    }
    if(ptrNodoActual == NULL){
      cout<<"El nodo con valor de dato "<<busqueda<<", no se encuentra en la lista.\n";
    }else{
      cout<<"Información del nodo encontrado:\n";
      cout<<"Direccion de memoria del nodo: "<<ptrNodoActual<<"\n";
      cout<<"Valor de (int dato): "<<ptrNodoActual->dato<<"\n";
      cout<<"valor de (nodo *ptrNodoSiguiente): "<<ptrNodoActual->ptrNodoSiguiente<<"\n\n";
    }
  }else{
    cout<<"La lista esta vacía.\n";
  }
  getch();
}
```

F

```
void modificar(){
  if(ptrNodoInicio != NULL && ptrNodoFinal != NULL){
    nodo *ptrNodoActual = ptrNodoInicio;
    int busqueda = 0;
    cout<<"Ingrese el dato a modificar:\n";
    cin>>busqueda;
    while(ptrNodoActual != NULL && ptrNodoActual->dato != busqueda){
      ptrNodoActual = ptrNodoActual->ptrNodoSiguiente;
    }
    if(ptrNodoActual == NULL){
      cout<<"El nodo con valor "<<busqueda<<", no se encuentra en la lista.\n";
    }else{
      cout<<"Ingrese el nuevo valor de (int dato) del nodo:\n";
      cin>>ptrNodoActual->dato;
    }
  }else{
    cout<<"La lista esta vacía.\n";
  }
  getch();
}
```

G

```
void imprimir_lista(){
  if(ptrNodoInicio != NULL && ptrNodoFinal != NULL){
    nodo *ptrNodoActual = ptrNodoInicio;
    while(ptrNodoActual != NULL){
      cout<<"Dirección de memoria del nodo: "<<ptrNodoActual<<"\n";
      cout<<"Valor de (int dato): "<<ptrNodoActual->dato<<"\n";
      cout<<"valor de (nodo *ptrNodoSiguiente): "<<ptrNodoActual->ptrNodoSiguiente<<"\n\n";
      ptrNodoActual = ptrNodoActual->ptrNodoSiguiente;
    {
  }else{
    cout<<"La lista esta vacía.\n";
  }
  getch();
}
```

H