Área Académica de Ingeniería en Computadores

CE3104-Lenguajes, compiladores e interpretes

BrailleRead

Realizado por:

Gabriel Chacón Alfaro, 2021049454

Emanuel Antonino Marín Gutiérrez, 2019067500

José Andrés Rodríguez Rojas, 2019279722

Nicol Otárola Porras, 20211117282

Profesor:

Marco Hernández

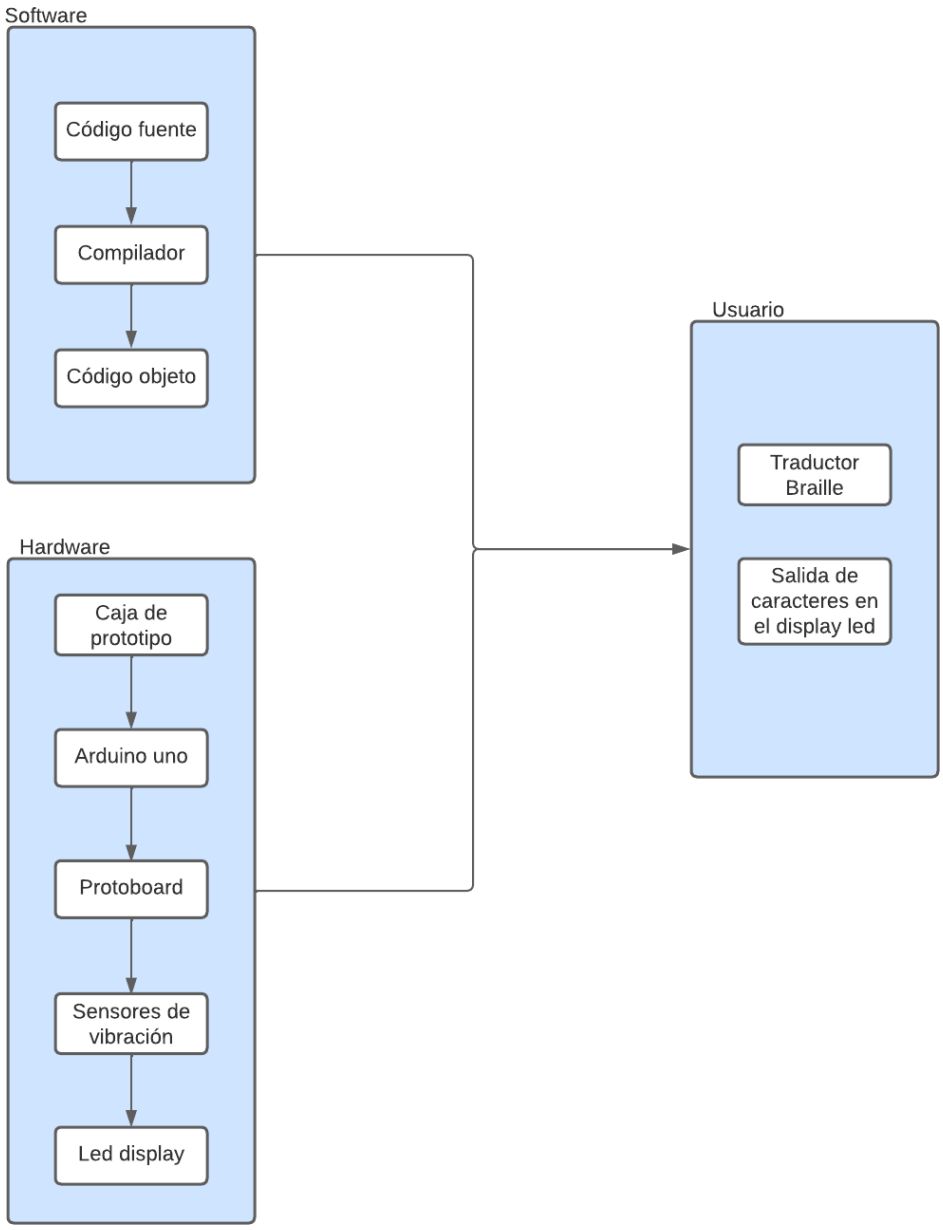
14 de junio, 2023

I Semestre

Contenido

Diagrama de arquitectura

Diagrama



Explicación

Código Fuente: Es el código escrito en un lenguaje de programación, en este caso BrailleRead.

Compilador: El compilador es responsable de tomar el código fuente como entrada y realizar una serie de pasos, como análisis léxico, análisis sintáctico y generación de código objeto. Su objetivo principal es traducir el código fuente en un formato comprensible.

Código Objeto: Es el código generado por el compilador en un formato específico para el Arduino, en este caso un string codificado.

Caja prototipo: Es una caja en la que se encuentran los componentes, el Arduino y la protoboard se encuentran en la parte inferior, mientras que la matriz led se encuentra en la parte de arriba en conjunto con los motores de vibración.

Arduino: El Arduino actúa como el controlador principal del sistema, recibe el código objeto y se encarga de ejecutar las instrucciones correspondientes.

Protoboard: Se conecta al arduino para que este posea el voltaje que van a necesitar los motores y la matriz led, que se encuentran conectados a la protoboard.

Motores de Vibración: Estos motores están conectados a la protoboard y son responsables de generar la vibración para simular el Braille. Cada conjunto de vibraciones puede representar un carácter o una combinación de caracteres Braille.

Matriz led: Están conectadas a la protoboard y muestra los caracteres que se van ejecutando en los motores.

Usuario con discapacidad visual: El usuario interactúa con el sistema, colocando sus dedos en los motores de vibración para recibir la información traducida al Braille.

Usuario: La pantalla muestra los caracteres para que los demás usuarios entiendan que es lo que los usuarios de braille están leyendo.

Problemas

Problemas conocidos

- Motores de vibración: Cuando el hardware se armó en su presentación final, se detectó que en el caso de que un motor vibre, la superficie sobre la que se encuentra también lo hace, lo que podría interferir con el tacto de los demás motores y se podría confundir cual es el motor que está vibrando realmente.

Problemas encontrados

- Análisis sintáctico y semántico: Durante el desarrollo del lenguaje de programación se llegó a confundir o mezclar partes del análisis sintáctico y semántico hasta el punto de que se habían definido errores de sintaxis cuando en realidad eran semánticos. Esto generó una regresión que implicó analizar con más detalle ambos tipos de análisis, en esto ayudó mucho el trabajo de investigación que se hizo con respecto al tema de Análisis Semántico en el Contexto de Compiladores y tomar el tiempo necesario para planear como implementarlos y de qué forma conectarlos. Para la creación del lenguaje y su compilador se utilizó como referencia la documentación de PLY (se recomienda leerla en su totalidad o gran parte de esta y hacer los ejemplos que vienen en las diferentes secciones) y conocimientos de Python a nivel intermedio.
- IDE: En cuanto a las fases iniciales, se encontraron varios inconvenientes en algunas funcionalidades del IDE. Uno de los principales problemas fue que se le querían agregar varias funciones al IDE para lograr que fuera más amigable con el usuario, pero en varias funcionalidades nos encontramos con la función "bind" de los widgets de Tkinter, la cual es muy limitada. Esta función "bind" nos permite capturar eventos que llaman a alguna función desde el widget, por ejemplo, el espacio de texto para codificar. En nuestro caso, cada vez que se apretaba una tecla en el espacio de texto para el código, se tenían que hacer varias funciones, por ejemplo, notificar los cambios al archivo o actualizar el número de línea, por lo que llamar a ambas funciones era imposible. Hacer una función combinada tampoco era viable ya que todas estas funciones también se utilizaban en otros eventos y recibían un parámetro de “event”. Posteriormente, para una solución no muy elegante, se repitió el código

de una función en otra para poder ejecutar ambas partes del código desde un mismo evento.

- Comunicación serial: Durante el proceso de pruebas (una vez que el hardware se encontraba en una fase funcional), se detectó un problema a la hora de enviar conjuntos de señales desde el lenguaje de programación hacia el Arduino, este problema consistía en una pérdida parcial de la información enviada al Arduino, tras realizar un análisis y una breve investigación, se encontró que la raíz del problema era la capacidad del buffer presente en el Arduino que se estaba utilizando (este es un problema que no es solucionable mediante software, ya que, en caso de requerir un buffer de mayor capacidad, la solución radica en comprar un dispositivo diferente o un Arduino distinto), ya que este buffer limitaba los mensajes que se recibían mediante el puerto serial a un máximo de 64 bytes, lo que provocaba que se perdiera el resto del mensaje recibido que superara los 64 bytes.
  Para solucionar dicho problema se optó por implementar una comunicación por segmentos de información ordenados, de modo que el lenguaje de programación envía segmentos de información de no más de 32 bytes al Arduino; en el caso de requerir enviar un mensaje cuyo tamaño supere los 32 bytes, este mensaje será fragmentado en subsegmentos de 32 bytes o menos, una vez realizada dicha fragmentación, el lenguaje de programación procederá a enviar ordenadamente las señales fragmentadas, esperando un tiempo prudencial entre cada envío. Cuando esto sucede, el Arduino se encarga de procesar las señales recibidas, llevar a cabo los procesos correspondientes, vaciar el buffer serial y, una vez que se encuentra listo para recibir más información, envía el mensaje “RTR” (Ready to receive) al lenguaje de programación y, cuando esto sucede, el lenguaje de programación procede a enviar el siguiente segmento de información, de este modo, se da una comunicación ordenada y en la cual, el buffer de comunicación serial presente en el Arduino, nunca se llega a saturar, evitando pérdidas de información y un malfuncionamiento del hardware.

Conclusiones

El análisis léxico al descomponer el texto en unidades más pequeñas llamadas tokens, como palabras, símbolos y números, y asignarles categorías gramaticales, se convierte en un proceso que es esencial para comprender y manipular el texto de manera eficiente y precisa.

Al implementar el análisis léxico, se logra una serie de beneficios, primeramente, permite identificar y eliminar los elementos irrelevantes o innecesarios del texto, como espacios en blanco, comentarios o caracteres especiales, lo cual simplifica el procesamiento posterior. Además, el análisis léxico ayuda a identificar palabras clave y símbolos específicos en el texto, lo que es fundamental para la construcción de sistemas de búsqueda y clasificación de textos.

Con el análisis sintáctico se logra descomponer las oraciones en sus componentes esenciales y establecer las relaciones entre ellos, lo que proporciona una base sólida para realizar tareas como la traducción automática, la generación de texto, la corrección gramatical y la extracción de información. Los beneficios que encontramos al implementar el análisis sintáctico incluyen una mayor precisión en la interpretación del lenguaje natural, una mejora en la comprensión del significado de las oraciones y una capacidad mejorada para realizar tareas de procesamiento de texto de manera automática.

Por otro lado, el análisis semántico es fundamental para comprender el significado y la intención detrás de un texto, este permite a las máquinas interpretar el contexto, las relaciones entre las palabras y las estructuras gramaticales, lo que resulta en una comprensión más profunda del contenido.

Al implementar el análisis semántico, se pueden lograr varios beneficios. Primero, ayuda a mejorar la precisión y la eficacia de los sistemas de procesamiento del lenguaje natural. Además, el análisis semántico es fundamental para la aplicación de tecnologías como la traducción automática, la generación de resúmenes, la clasificación de documentos y la extracción de información. Al comprender el significado real detrás de un texto, las máquinas pueden realizar tareas más complejas y sofisticadas, lo que mejora la experiencia del usuario.

En la traducción se involucran tanto el análisis sintáctico como el análisis semántico para descomponer y entender las estructuras gramaticales de las oraciones en el idioma fuente y

luego generar oraciones equivalentes en el idioma objetivo. Por lo anterior, es que el proceso de traducción es fundamental para convertir el código fuente en un programa ejecutable.

Finalmente, el construir un traductor Braille en Arduino y una serie de motores, es un proyecto enriquecedor que combina tecnología, accesibilidad y empoderamiento, además, proporciona una herramienta útil para facilitar la comunicación y el acceso a la información para las personas con discapacidad visual, al tiempo que promueve el aprendizaje y el desarrollo de habilidades técnicas.

Recomendaciones

Que se aprovechen las herramientas y bibliotecas existentes que les pueda simplificar la implementación del análisis léxico en código, un ejemplo es implementar el Lex. Además, es esencial diseñar una estrategia adecuada y establecer reglas claras para el reconocimiento de tokens relevantes en el texto, esto permitirá comprender y manipular el texto de manera eficiente y precisa en tu aplicación.

Seleccionar una biblioteca o herramienta de análisis sintáctico que se adapte a tus necesidades, por ejemplo, se puede utilizar, Yacc. Asimismo, es necesario realizar un preprocesamiento del texto, que incluya eliminar signos de puntuación, tokenización, lematización y eliminación de palabras irrelevantes. Realizar pruebas e ir ajustando tu implementación, realizando pruebas exhaustivas con diferentes tipos de oraciones para determinar que podría hacer que el programa falle y con esto, considerar posibles mejoras en el preprocesamiento o los parámetros de la función de análisis sintáctico.

Se debe considerar la utilización de técnicas de aprendizaje automático para mejorar la capacidad de análisis semántico de tu sistema, esto debido a que el análisis semántico es un campo en constante evolución, por lo que es importante mantenerse actualizado con los avances y las nuevas técnicas que surjan. Además, es recomendable realizar pruebas exhaustivas y ajustes en tu implementación para garantizar la precisión y la eficacia en la comprensión del significado de los textos, por último, no olvides considerar las necesidades y los requisitos específicos de tu proyecto al elegir las herramientas y técnicas de análisis semántico. Cada proyecto puede requerir enfoques y modelos diferentes según el dominio y los objetivos específicos que se deseen alcanzar.

Tomar en cuenta que se utiliza la información obtenida del análisis sintáctico y semántico para generar el código equivalente en el idioma objetivo, para ello, es recomendable, aplicar las reglas y convenciones del lenguaje objetivo. Después de generar el código objetivo, es fundamental realizar pruebas unitarias y pruebas de integración para verificar que el programa traducido se ejecute según lo esperado.

Asegúrate de realizar y mantener el orden mientras se está montando el hardware, en la medida de lo posible, realice pruebas con personas con discapacidad visual para obtener retroalimentación y realizar ajustes que mejoren la experiencia de uso.

Referencias

ASCII-Braille Real-Time Translation via Arduino. (s/f). projecthub.arduino.cc. Recuperado el 14 de junio de 2023, de https://projecthub.arduino.cc/CesareBrizio/ascii-braille-real-time-translation-via-arduino-3c90dd

Interfaces gráficas de usuario con Tk. (2023). Python Documentation. https://docs.python.org/es/3/library/tk.html

Lumunge, E. (2022, enero 7). *Semantic analysis in compiler design*. OpenGenus IQ: Computing Expertise & Legacy. https://iq.opengenus.org/semantic-analysis-in-compiler-design/

PLY (python Lex-Yacc). (s/f). Dabeaz.com. Recuperado el 15 de mayo de 2023, de https://www.dabeaz.com/ply/