Instituto Tecnológico de Costa Rica.

Escuela de Ingeniería en Computación.

Compiladores e Intérpretes.

Prof. Francisco Torres Rojas.

Apuntes del Viernes 03 de Marzo del 2017.

Estudiante:

Ximena de los Ángeles Bolaños Fonseca.

2015073844

I SEMESTRE

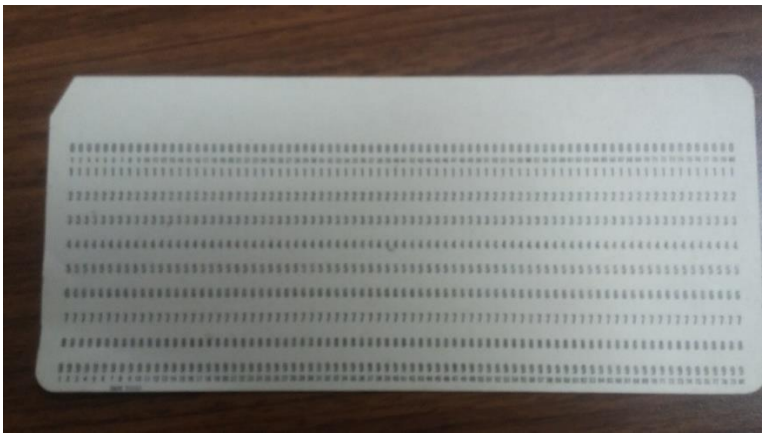
Contenido

Aspectos Administrativos 3
"Parientes" de los Compiladores 4
Programas relacionados con los Compiladores 4
Ensambladores 5
Desensambladores 7
Intérpretes. 10
Preprocesadores. 10
Linker. 11
Loader 12
Debugger 13
Profilier 13
Editor de Texto 15
Manejadores de Versiones 16
Estructura Básica de un Compilador 17
Entradas y Salidas 17
Representación Intermedia 17
Análisis y Síntesis 18
Ejemplo 18

Aspectos Administrativos

- El valor del proyecto será de 6%, de momento.
- Dependiendo de la cantidad de proyectos, ese 35% se dividirá proporcionalmente.
- Posiblemente el fin de semana se deja la 2da Tarea Programada.
- Investigar sobre Flex y Beamer (presentaciones en Látex)
- Nadie ha perdido el curso, entonces tranquilos.
- Se dio la entrega del Quiz 3, la segunda Tarea y la evaluación del Proyecto 0.

Tarjetas Perforadas



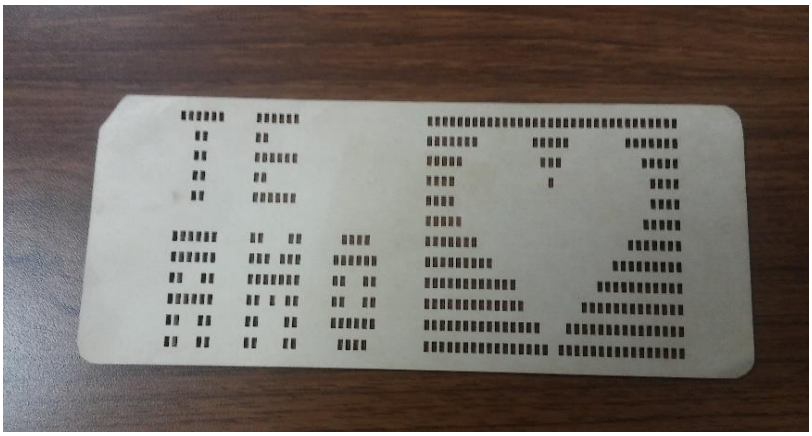
Matrículas Con las Tarjetas Perforadas.

Primera Matrícula del Profe en la U.

Era cuando aún estaba en física, pero el papá lo convenció de que si estudiaba física iba a ser profesor.

Regalo de una novia del Profe.

*Awww :3 y, ¿con eso ya se casaba uno? No, solo tenía sexo salvaje.



"Parientes" de los Compiladores

Programas relacionados con los Compiladores

El software siempre está relacionado directa o indirectamente con los compiladores. Los Sistemas Operativos u los lenguajes de Alto Nivel han sido sumamente exitosos. Desarrollo de software actual totalmente asociado a su uso. Tecnologías que influencian programadores.

Ensambladores

- Inventados en los 1950's.
- Todavía viven, no se utiliza mucho pero aún están.
- Programación de bajo nivel.
- Relacionado 1 a 1 con el lenguaje máquina, no se pierde nada de eficiencia.
- Codificación Eficiente.
- Sintaxis muy simple, en comparación a uno de alto nivel.

¿Alguna vez los han amenazado con hacer un Ensamblador?

Hacerlo no es tan difícil. Todos son iguales… igual de feos. Es muy estructurado:

- Campo de Etiquetas ¿Está o no está?
- Código de Operación: Bien escrito o mal escrito
- Argumentos

A la hora de la definición de espacios, en el archivo de salida se deja el campo para la definición de los campos.

¿Por qué se llaman ensambladores?

Porque usted lo que hace es ensamblar cada instrucción que va a salir.

Campos fijos (facilidad para el programador):

- Etiquetas.
- Código de Operación.
- Argumentos.

Usan técnicas de compilación. Normalmente se realiza en dos pasadas **(Salen en el Libro de la Tarea):**

- La creación de la tabla de símbolos. Se define en qué posición esta la etiqueta y se guarda en la tabla de símbolos.

  Ejemplo:

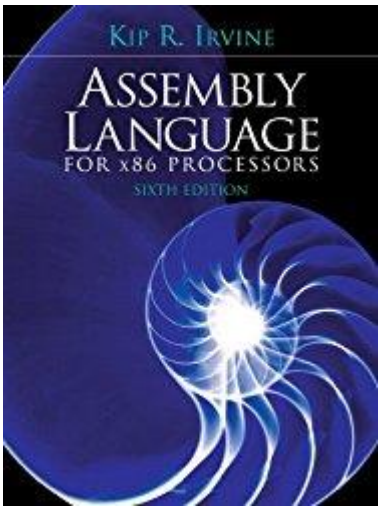
```
            jmp ACA
            ….
            ….
ACA:
            ….
            ….
```

En la tabla de símbolos se escribe:

ACA con 417 bytes.

macros se guarda la hilera

- Hago el código, escupe código máquina:
  jmp encontrar el código máquina FC en la posición 417.



¿Es posible hacer un compilador que sea más bien a alto nivel?

Sí, un compilador es software que traduce un programa escrito en el lenguaje de una máquina virtual a un programa equivalente escrito en otra maquina virtual. Por ejemplo: Forchan (No vino :() a C.

¿Es posible pasar de un lenguaje interpretado a uno compilado?

Depende. Porque el interpretado que ocupa el ambiente completo. Por ejemplo: LISP pedazo de datos que se pueden correr como un programa.

¿Ensamblador de una pasada? Sí se puede, pero de la otra manera es menos complicado.

- Va leyendo
- Va escupiendo
- Va armando la tabla de símbolos

FC 4217. No esta jmp en la tabla de Símbolos. ¿Qué hago?

- Busco ACA, no está, creo una entrada.
- Lista enlazada de símbolos.
- Luego aparece ACA se devuelve Código Objeto.
- Lugares que faltaban se arreglan(“Backpatching”).

Para escribir compiladores hay que saber bastante de ensamblador. Para optimizar el código hay que ser monstruos peludos en ensamblador.

Antes todos los ensambladores eran similares. La arquitectura siempre era la que mandaba.

Desensambladores

Es volver al código ensamblador. Sigue el proceso inverso de un ensamblador. A la hora de las etiquetas es complicado es por ello que se manejan como los temporales que se utilizaron en el proyecto 0.

Ejemplo: jmp 414, Entonces se crea una etiqueta en esa posición para que el humano pueda entender. Se tiene que saber siempre en qué posición están.

Este siempre toma un archivo de lenguaje máquina y lo muestra en ensamblador. Es como un simulador. Debe conocer las formaciones en binario. Reconoce el código operación y las formas de direccionamiento. Genera etiquetas y nombres de variables arbitrarias.

¿Una Pasada? Este sería de alguna manera como el “backpatching”.

¿Dos Pasadas? Se daría un posible uso de etiquetas. En donde en la primera pasada ubica la cantidad de memoria a la que salta el jmp y en la segunda asigna las etiquetas donde corresponde.

Las macros siempre se van a perder. Solo si se es muy “pro” se crea un ensamblador que sepa que si se repite mucho una porción de código cree una macro, pero eso sería *“Gastar pólvora en zopilotes”.*

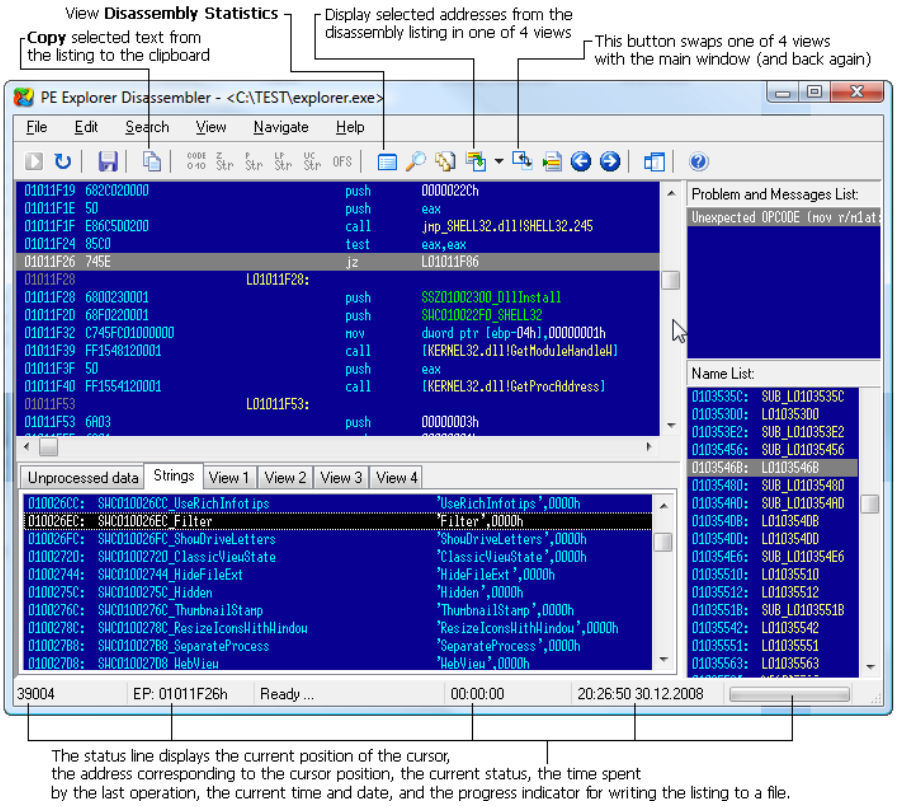
¿Datos vs el Código? Es complicado, con los define bytes**.** Tiene que ser muy carga. Si tienen sus retos.

¿Macro? ¿Pseudoinstrucciones? Se tiene que ser muy pro como se dijo antes para poder generarlas, pero normalmente se pierden. Ya que es azúcar sintáctico que hace el programador súper feliz.

Se reconoce por una red.

¿Utilidad?

Ingeniería inversa, Antivirus, Debuggers o Códigos viejos en donde se ha perdido la fuente. (device drivers)



Recomendación

Canal AMC, la serie "Call and Catch fire": historias de la computación de los 80. En una parte sale cuando se roban que el SO de IBM PC con un desensamblador.

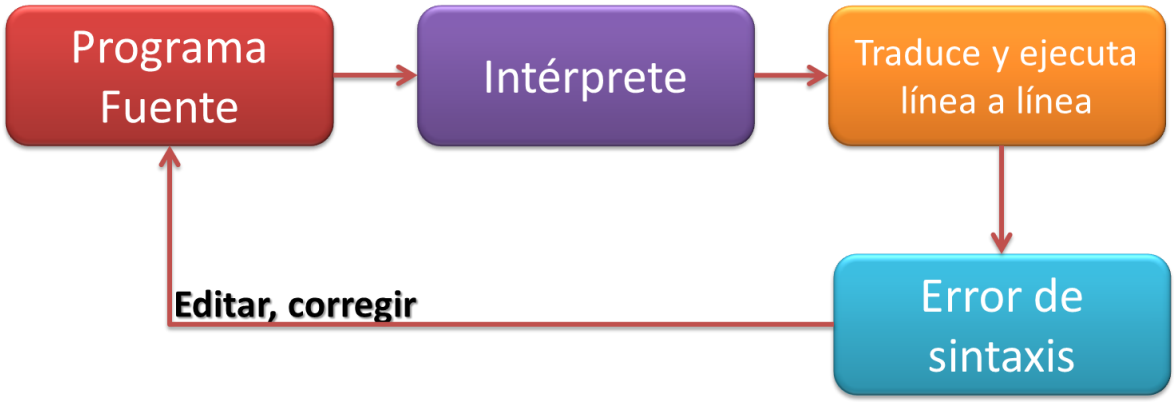
Intérpretes.

Película: The Interpreter. Con Nicole Kidman.

Combinación de la traducción con la ejecución. Corre línea por línea. Utiliza muchas técnicas equivalentes a las de compiladores. Una de sus pocas virtudes, que es bastante independiente de la arquitectura. Funciona como “Run-Time enviroment”, prácticamente Máquinas Virtuales. Son Bastante lentos en ejecución, ya que todo se repite cada vez.

Ejemplos: LISP, BASIC, Python, Ruby y Forth (buenos en punteros, por dicha ya no se utiliza 😊)

¿Aunque tenga errores de sintaxis y sigue corriendo es bueno? NO.



Preprocesadores.

Muy obvio lo que hace, ¿no? Se ejecuta antes del proceso. En C se confunde a la hora de compilar, son dos procesos aparte, pero siempre se realiza antes de la compilación. Son muy útiles. Como, por ejemplo: #pragma para que C siga siento portátil. Como se dijo antes ocurren siempre antes de la compilación, como por ejemplos los includes.

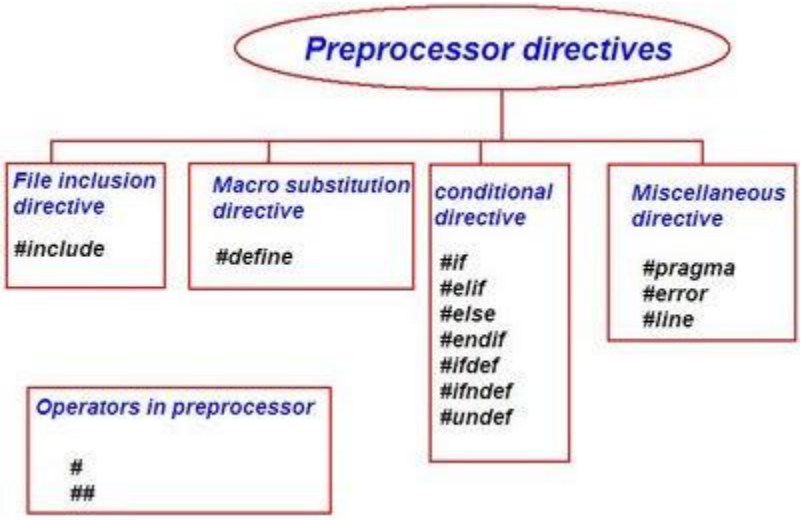
Entre sus funciones típicas están:

- Incluir archivos
- Reemplazo de hileras
- Macros. Se definen y luego en algún lugar se usa.
- Eliminación de Comentarios.
- Compilación condicional (que se adopte un montón de fuente #if o #else).

Buen estilo de programación porque si en una corrida una función está en la posición 17, luego si se mueve no va a estar en la 17 por lo tanto se recompila para que se pueda saber dónde está y si está en la 24 se busca en la posición 24.

Usan técnicas de compilación, es casi como un lenguaje. Antes en C se utilizaba como un programa aparte. Hay que tener claro que, aunque sea de C no es C, aunque antes se pensaba que por saber hacer un preprocesador se sabía del lenguaje no necesariamente. Además, en la actualidad en C se ve como un proceso transparente.

Con el ejemplo anterior, se reitera que es independiente del lenguaje.



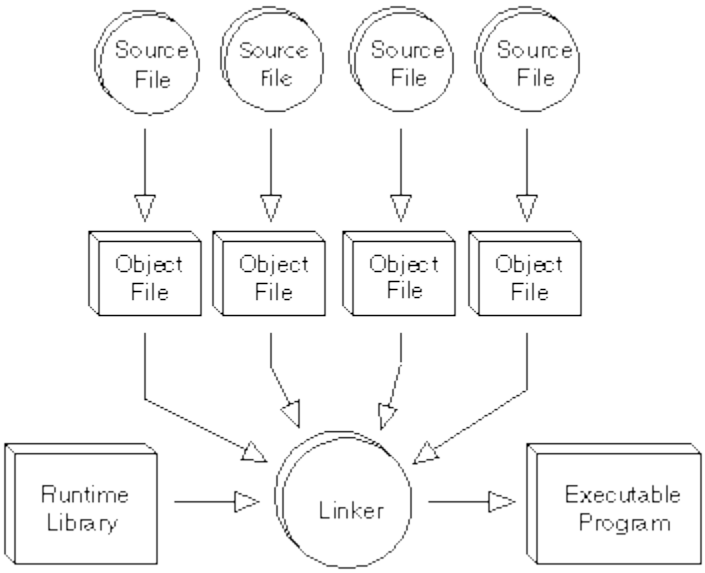
Linker.

(una manita de gato) No se hablará en gran profundidad porque para eso está la TAREA que es para el próximo miércoles 8 de marzo.

Esto es como el manejo de Bibliotecas. Por ejemplo: printf no es parte del lenguaje original de C, por lo que hay que pegarlo al ejecutable.

Su traducción sería “Ligador” o “Eslabonador” Los compiladores o ensambladores ganaran módulo objeto. Los programas o funciones compilados independientemente. Son como las bibliotecas con funciones. Se crean módulos ejecutables (Ejemplo: el .exe en Windows, GCC) Muchas veces viene integrado con el compilador. Existen dos tipos:

1. Linking estático: Todo lo que se podría necesitar ya se encuentra en el archivo, quita muchas memorias eso sí, es más rápido, están separados y siempre van a correr.
2. Linking Dinámico: Se pone en una función. Necesita que estén las bibliotecas por lo que siempre estén en mejora a la hora de llamar una función. Utilizan menos memoria. La mayoría de los ejecutables lo usan. Un ejemplo sería Actualización del DLL.



Sugiere más un Linking estático esta imagen.

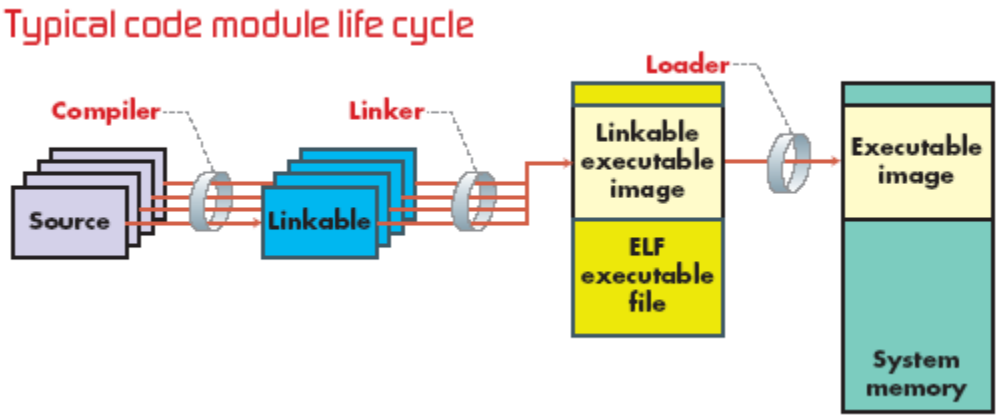
**Se me esta yendo la voz… es por la oscuridad.

Loader

Hubo una época en la que estuvo aparte. Su traducción sería cargador. Los .exe en la actualidad los cargan a memoria el loader. Toma un módulo de ejecución y lo carga a memoria. Relocaliza. Es parte del Sistema Operativo. Se debe de está negociando el espacio. Crea estructuras de control. Ejemplo: los PCB (Process Control Block) es lo que el sistema operativo manipula a la hora de los procesos en cola. Este lo crea el loader. Cada sistema operativo dentro de eso es genérico.

En 1940 todo esto eran piezas. Pero es mejor no retroceder.

El loader en la actualidad es muy transparente. Como por ejemplo cuando llegan los mensajes de WhatsApp y uno no se da ni cuenta que hay un loader detrás de eso.



Debugger

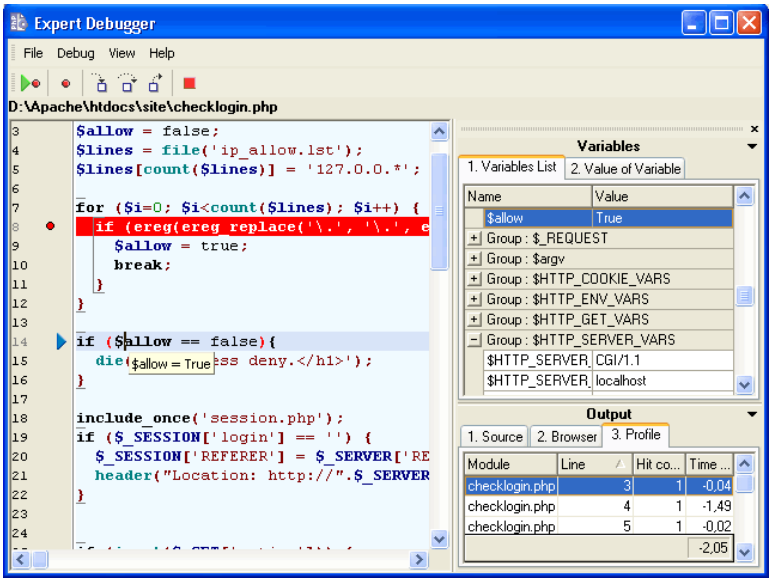
Realmente no es indispensable.

Regla: USTEDES NO TIENEN DERECHO A USAR UNA HERRAMIENTA SI NO SABE COMO SE USA. SI NO ES MAGIA NEGRA.

Es una alcahuetería, una belleza. Permiten la ejecución en forma controlada de otro programa. Genera instrucción por instrucción. Se maneja en flujo de ejecución. Cuando hay un Breakpoint (hardware) se reactiva el debugger. ¿Cómo sé que el contenido de las variables? Con magia negra… Realmente el compilador le ayuda, este deja mucha información de control en la tabla de símbolos este lo exporta al debugger cada vez que hay un call.

El problema es que deja mucha basura, como CodeBlocks. Por lo que el programa queda con mucha Manteca (programa fuente disponible).

Hace al programador más vagabundo.

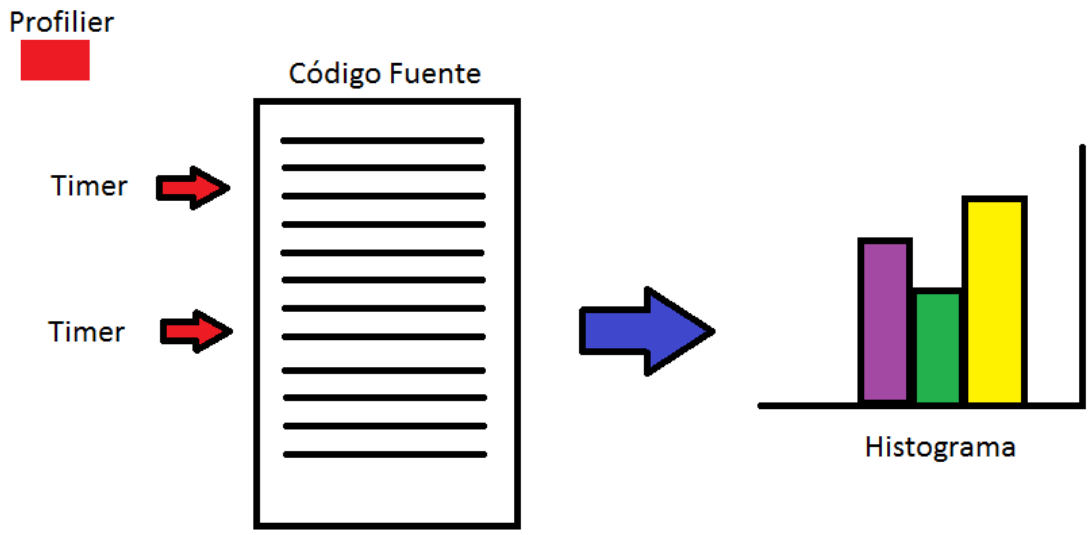


Profilier

Programa que recolecta estadísticas de la ejecución de otro programa (¿Por dónde paso? Números de veces que se llamó, Numero de tiempo promedios). Es el perfil de un programa. Ayuda a mejorar el rendimiento de un programa, que se puede ver en que función se dura más. Muy útil para mejorar la calidad del programa

analizado. En el caso que se quiera optimizar una función esta se puede reescribir en ensamblador para mejorar su diseño.

El profilier necesita información creada por el compilador (Más manteca). Cual función corresponde a una función aleatoria del programa.

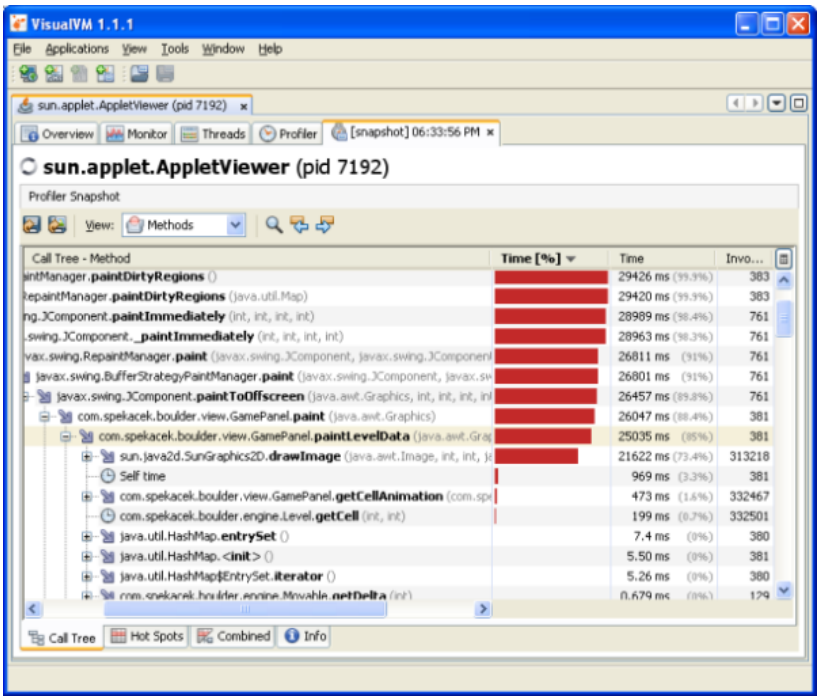


En esta Imagen se muestra como el Profilier cada cierto tiempo recolecta la información, hace un muestreo estadístico, tratando de ser lo menos intrusivo al programa. Finalmente generando un Histograma.

El compilador se puede beneficiar de los datos recolectados por el profilier, por si se generó código cochino. Se hace en un ambiente súper automático. Como por ejemplo los celulares Android.

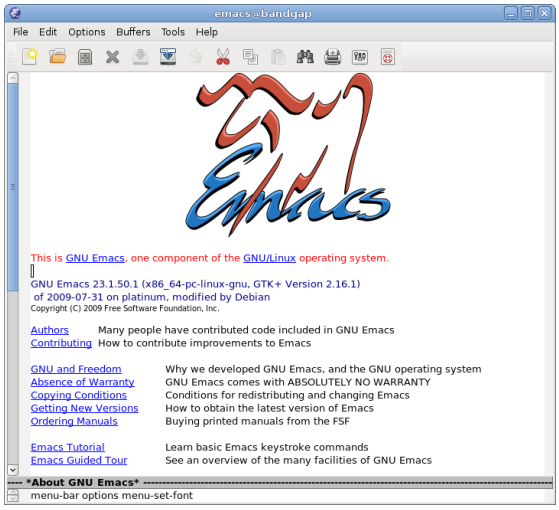
Técnicas más agresivas de optimización para los puntos calientes del código. La recopilación se hace automática. No siempre va a estar (En SQL se utiliza, pero solo si el programador se da cuenta). Muchas veces el profilier a uno le enseña la función en la que menos uno se espera que va a estar el problema.

Como por ejemplo que se hizo una función con Burbuja y pasándola a QuickSort funciona.



Editor de Texto

En este caso es de ayuda para el programador o bien el programador le ayuda al editor (mínimo al nivel de técnicas) y sabe mucho del lenguaje. Es como los compiladores que contienen un buscador de contador y el manejo de expresiones regulares. Muestra cuando una variable está declarada o no. Ejemplo: EMACS creado por Stallman, manejo de plugins, sabe que es látex, hasta cuando le hace una carta a la novia.



Los programas fuentes son escritos con editores de texto. Usan técnicas de compiladores para sus funciones, como las tablas de símbolos. Manejo de Expresiones regulares para búsquedas. El código es cargado para alterar el

comportamiento (plugins). Editor con propósito general orientado a un leguaje. Ejemplo: PDF (Portal Document Format) es un lenguaje por dentro. Los Editores tienen el conocimiento de la sintaxis de un lenguaje. Genera partes, marca con colores o con algún tipo de letra especial. Informa de los errores sintácticos. Combinación de editores, combinación de profiliers y de debugguers.

Integrates Development Enviroment(IDE)

Manejadores de Versiones

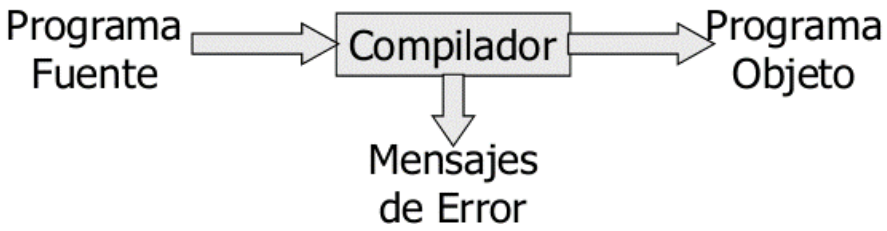
Normalmente se utilizan cuando son los proyectos grandes de software que requieren de mucha gente. Se trabaja siempre sobre la misma fuente. Comentarios de las versiones de la fuente. En los documentos se sabe que hizo quien y cuando. Se puede regresar a puntos previos del desarrollo. Es totalmente independiente del lenguaje o requerir el conocimiento sintáctico del lenguaje para registrar las cosas.

Ejemplo: GIT, Creado por Linus Torvalds.

Estructura Básica de un Compilador

Entradas y Salidas

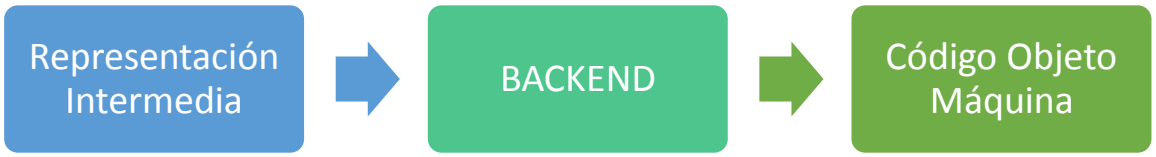
- Entrada: lenguaje fuente.
- Salida: lenguaje objeto.



Representación Intermedia

Compuesto de dos fases: FrontEnd y el BackEnd.

Como se puede denotar en el FrontEnd no importa la salida o en lo que va a acabar, solo el código Fuente.



Por el contrario, en el BackEnd sí importa la salida, pero no importa el código fuente.

Análisis y Síntesis

- Análisis: tomar algo y descomponerlo en partes para entenderlo.
- Síntesis: construcción de algo a partir de diferentes partes.

Entonces se combina en:

- Front End: Análisis
- Back End: Síntesis

Entre más independientes sean estas fases más flexibilidad hay, lo que hace un buen compilador.

Ejemplo:

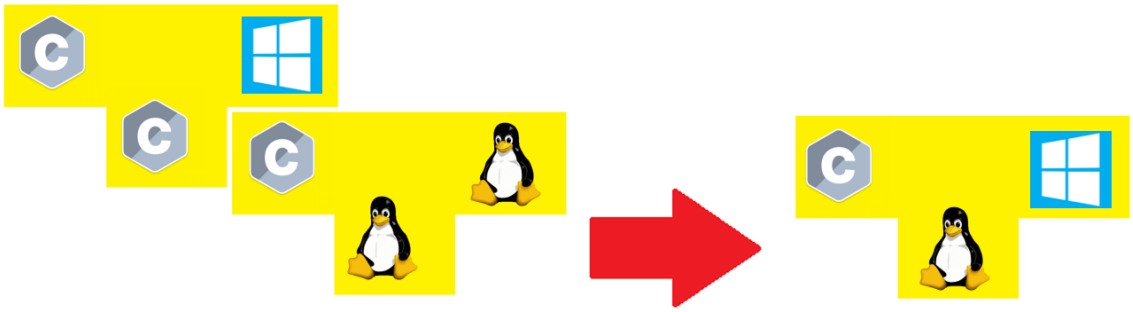
- Hay que desarrollar en Linux un Compilador de C.
- Este Compilador se va a ejecutar en Windows.
- Se va a generar un Código que corre en Windows.

- Tenemos el compilador GCC que ejecuta Linux y genera Linux.
- Tenemos el fuente del compilador GCC
- Podemos estudiar el fuente e identificar el BackEnd (En la tarea programada es lo que se manejó en su mayoría).
- Una vez estudiado el BackEnd.
- Se modifica el código para que genere el código Windows.

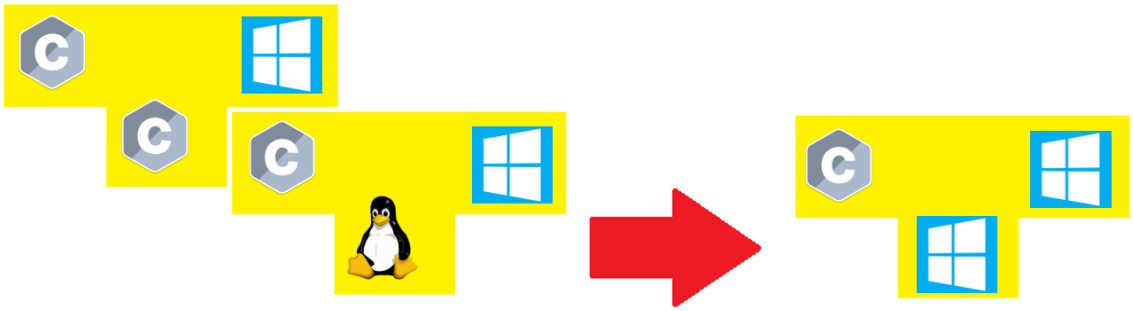
¿Difícil?

Hay que identificar el BackEnd lo cual no es trivial, pero si factible. Stallman lo hizo. No toco para nada el análisis solo la síntesis.

- Compilador con GCC en Linux



- Compilar con un nuevo compilador en Linux



Una vez finalizado: **¡NO le diga a Nadie que lo hice!**

La Separación hizo que fuera más flexible.