INSTITUTO TECNOLÓGICO DE NUEVO LEÓN

Ingeniero en sistemas computacionales

Lenguajes Autómatas II

Proyecto #3

Catedrático

Juan Pablo Rosas Baldazo

Alumno

Carlos Alberto Martínez Mendoza

Fecha

16 de Abril del 2018

Introducción

En esta unidad conoceremos lo que es la optimización dentro del código (ya sea el código intermedio o el código objeto) y como esta fase tiene mucha utilidad al momento de compilar nuestro código.

También conoceremos los diferentes tipos de optimizaciones que existen y cuál es su función principal por si queremos utilizar alguno de estos. Sin dejar en claro cuáles son sus ventajas y desventajas de utilizarlo.

Otra cosa que veremos en este resumen de esta unidad será el de los costes dentro de nuestros equipos de cómputo y como poder sacarle el máximo provecho a nuestra máquina. Cabe resaltar que también conoceremos todas las herramientas y su modo de uso para el análisis de flujo de datos.

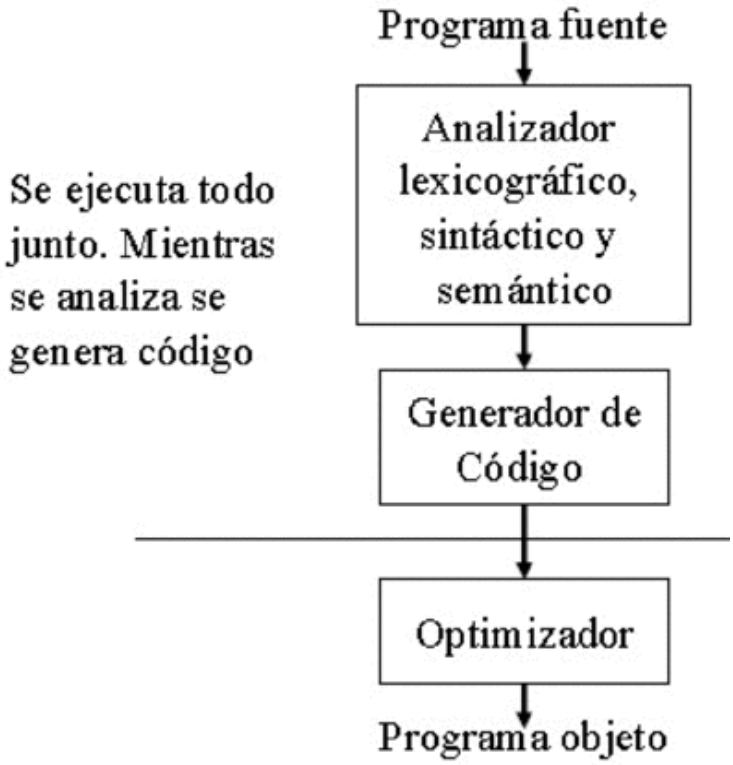
Por último, veremos qué criterios debemos tomar en cuenta al momento de querer mejorar nuestro código y cómo podemos optimizarlo de manera que no lleguemos a utilizar demasiada memoria o más recursos.

Unidad 3: Optimización

El objetivo de esta unidad aplicado en la práctica será:

- Obtener código que se ejecuta más eficientemente según los criterios
- Tiempo de ejecución (optimización temporal)
- Espacio de memoria utilizado (optimización)

La fase de optimización de código consiste en mejorar el código intermedio, de modo que resulte un código máquina más rápido de ejecutar



Su funcionamiento consiste en revisar el código generado a varios niveles de abstracción y realiza las optimizaciones aplicables al nivel de abstracción. Por medio de esto, el mismo código lo representa desde el más abstracto hasta el menos abstracto

Capítulo 1: Tipos de optimización

La optimización va a depender del lenguaje de programación y es directamente proporcional al tiempo de compilación; es decir, entre más optimización, más tiempo de

compilación. Algunos editores ofrecen una versión de depuración y otra de entrega o final.

La optimización es un proceso que tiene a minimizar o maximizar alguna variable de rendimiento, generalmente tiempo, espacio, procesador, entre otros. Desafortunadamente no existen optimizador que hagan un programa más rápido y que ocupe menor espacio.

En si la mayoría de los tipos de optimizaciones sirven para diferentes funciones, tales son:

- Al momento de la compilación
- Cuando es independiente de la maquina
- Un transformador en código intermedio
- Un apoyo en reducción de coste, expresiones comunes, propagación de constantes
- Cuando depende de la misma maquina
- Asignación de registros
- Ordena y selecciona las instrucciones

Sección 1.1: Locales

La optimización local se realiza sobre módulos del programa. En la mayoría de las ocasiones a través de funciones, métodos, procedimientos, clases, entre otras.

La característica principal de este tipo de optimización es que solo se ven reflejadas en algunas secciones, ya que solo sirve cuando un bloque de programa o sección es crítico (entre más pequeño sea el espacio de soluciones, la optimización local será más rápida)

Folding (Ensamblamiento)

Este consiste en reemplazar las expresiones por su resultado cuando se pueden evaluar en tiempo de compilación (resultado constante). Permite que el programador utilice cálculos entre constantes representados explícitamente sin introducir ineficiencias

Ejemplo de Folding

Expresión: `3-(5+6)-A*10+4`

Árbol:

```
-
+- +
|   +- -
|   |   +- 3
|   |   +- +
|   |       +- 5
|   |       +- 6
|   +- *
|       +- A
|       +- 10
+- 4
```

Términos:

3
-5
-6
-(A*10)
4

Términos constantes:

3-5-6+4 = -4

Resultado: `-4-(A*10)`

Propagación de constantes

Desde que se asigna a una variable un valor constante hasta la siguiente asignación, se considera a la variable equivalente a la constante.

Ejemplo: PI=3.14 -> PI=3.14 -> PI=3.14

G2R=PI/180 -> G2R=3.14/180 -> G2R=0.017

PI y G2R se consideran constantes hasta la próxima asignación.

Estas optimizaciones permiten que el programador utilice variables como constantes sin introducir ineficiencias.

Implementación de la propagación de constantes

- **Separar el árbol en bloques básicos**

Cada bloque básico será una lista de expresiones y asignaciones

- **Para cada bloque básico**

Inicializar el conjunto de definiciones a conjunto vacío.

• Definición: (variable, constante)

Procesar secuencialmente la lista de expresiones y asignaciones

- **Para expresión y asignación**

Sustituir las apariciones de las variables que se encuentran en el conjunto de definiciones por sus constantes asociadas.

- **Para asignaciones**

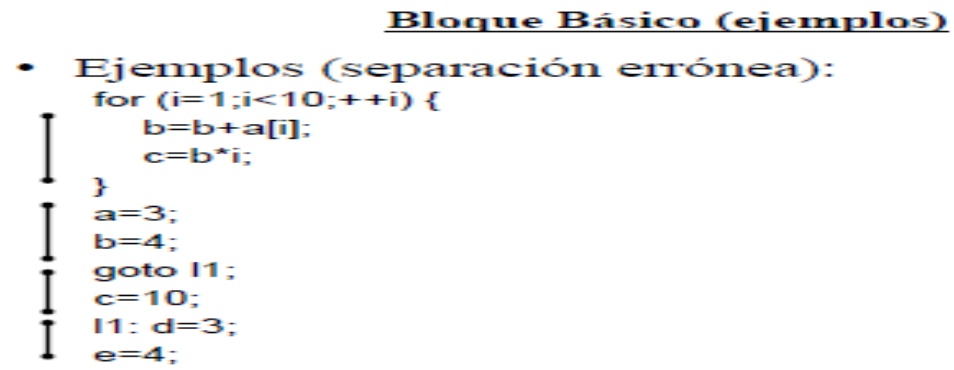
Eliminar del conjunto de definiciones la definición de la variable asignada

Añadir la definición de la variable asignada si se le asigna una constante.

Bloque básico

Un bloque básico es un fragmento de código que tiene una única entrada y salida, y cuyas instrucciones se ejecutan secuencialmente.

La idea del bloque básico es encontrar partes del programa cuyo análisis necesario para la optimización sea lo más simple posible.



Bloque Básico (ejemplos)

- Separación correcta

```
for (i=1;i<10;++i) {
    b=b+a[i];
    c=b*i;
}
a=3;
b=4;
goto l1;
c=10;
l1: d=3;
e=4;
```

```
BB1:
        i=1;
BB2:
        i<10;
BB3:
        b=b+a[i];
        c=b*i;
        ++i
BB4:
        a=3;
        b=4;
        goto l1;
BB5:
        c=10;
BB6:
        l1: d=3;
        e=4;
```

Reducción de potencias

Se busca sustituir operaciones costosas por otras más simples.

Ejemplo:

– sustituir productos por sumas.

a=2*a

a= a + a

– Evitar la operación (++)

A=length (s1 ++ s2)

Convertirlo en A=length(s1)+length(s2)

Sección 1.2: Ciclos

Los ciclos son una de las partes más esenciales en el rendimiento de un programa dado que realizan acciones repetitivas, y si dichas acciones están mal realizadas, el problema se hace N veces más grandes. La mayoría de las optimizaciones sobre ciclos tratan de encontrar elementos que no deben repetirse en un ciclo.

El problema de la optimización en ciclos y en general radica en que es muy difícil saber el uso exacto de algunas instrucciones. Así que no todo código de proceso puede ser optimizado. Otro uso de la optimización puede ser el mejoramiento de consultas en SQL o en aplicaciones remotas (sockets, E/S, etc.).

- Sea el ejemplo:

```
while(a == b) {
  int c = a;
  c = 5;
  …;
}
```

- En este caso es mejor pasar el int c =a; fuera del ciclo de ser posible.

Invariante del ciclo

No tiene sentido evaluarla dentro del ciclo ya que se mueve el código fuera del ciclo. Su destino del código es el bloque de entrada al ciclo, i es una variable de inducción para un ciclo L si existe una constante c tal que cada vez que se asigna un valor a i, su valor aumenta c – note que c puede ser negativa o positiva

Sección 1.3: Globales

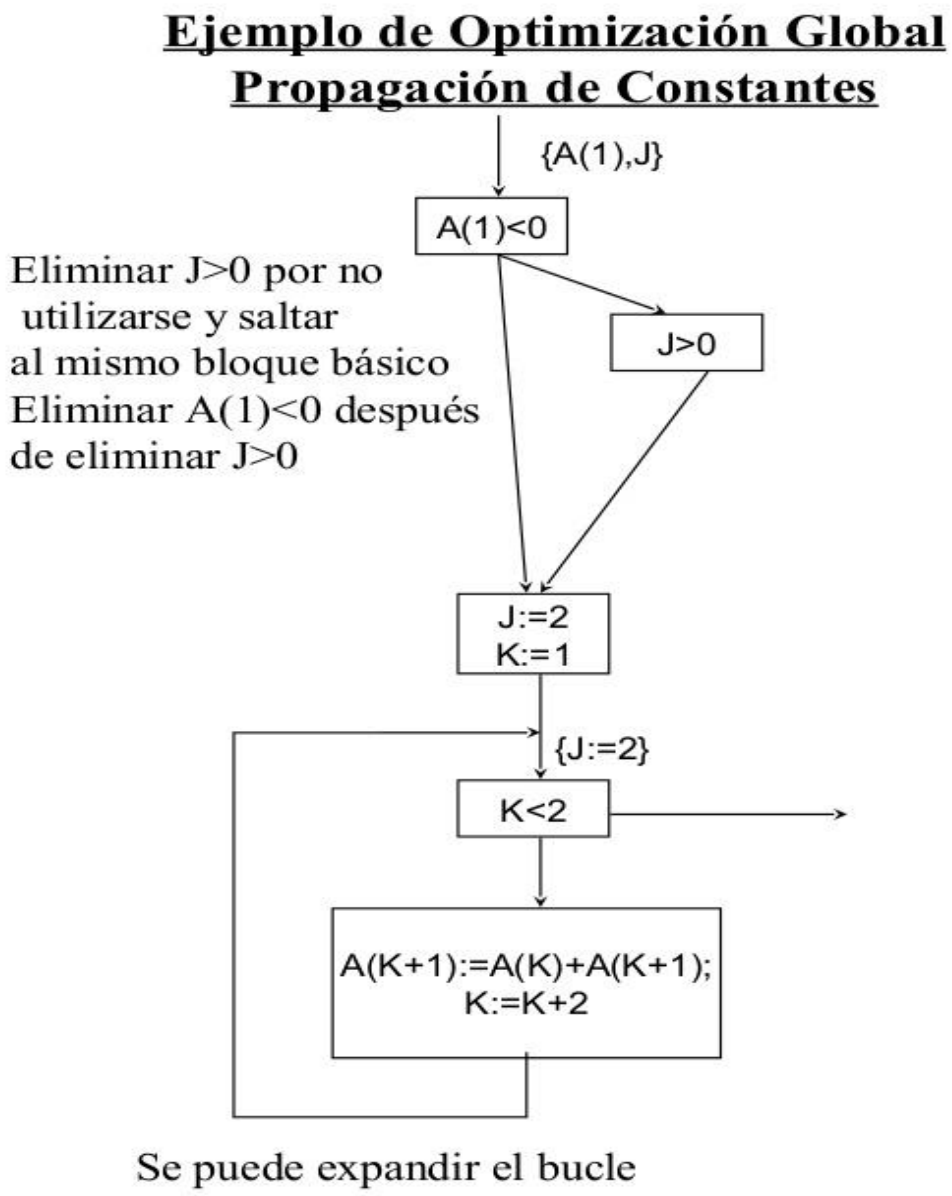
La optimización global se da con respecto a todo el código. Este tipo de optimización es más lenta pero mejora el desempeño general de todo programa.

Las optimizaciones globales pueden depender de la arquitectura de la máquina. En algunos casos es mejor mantener variables globales para agilizar los procesos (el proceso de declarar variables y eliminarlas toma su tiempo) pero consume más memoria.

Algunas optimizaciones incluyen utilizar como variables registros del CPU, utilizar instrucciones en ensamblador.

Tipos de optimización global

- Estocásticos: Evalúan f sobre una muestra aleatoria sobre la religión de interés. Sin embargo, la eficiencia es una característica de estos métodos ya que a problemas de gran escala (+100 variables) son resueltos mejor con estos métodos
- Determinísticos: No envuelve ningún elemento aleatorio de forma más confiable por medio de métodos de puntos. Estos métodos calculan valores de la función sobre puntos de muestras (incapaces de resolver confiablemente un problema de optimización global). Otro método también es el método de bordes, este ya mencionado consiste en calcular cotas sobre conjuntos compactos. Si son implementados apropiadamente y considerando errores de redondeo, pueden producir soluciones globales rigurosas



Grafo del flujo de ejecución

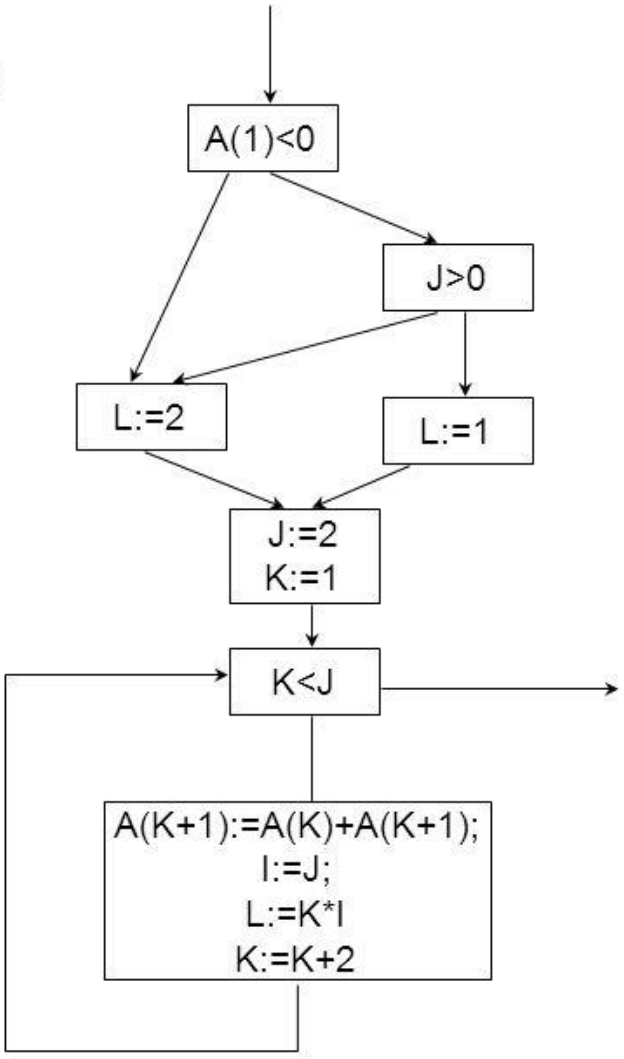
Antes de realizar una optimización global es necesario crear el grafo de flujo de ejecución. Este grafo representa todos los caminos posibles de ejecución del programa

Por medio del análisis del grafo del flujo para la optimización global, permitirá al código:

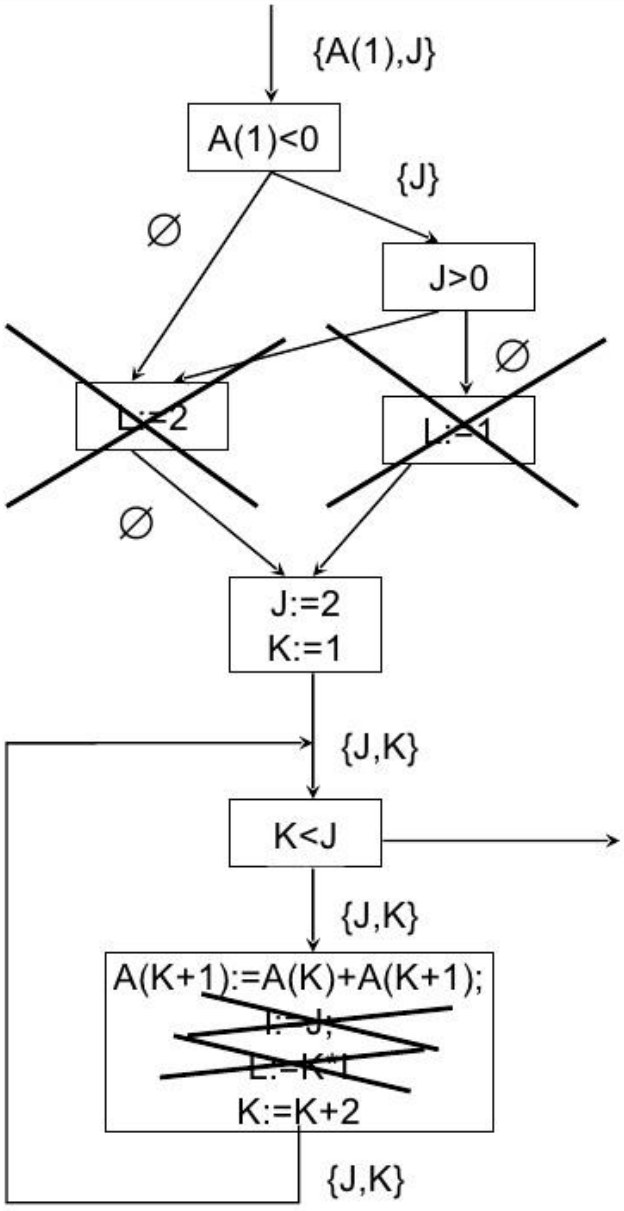
- Una propagación de constantes fuera del bloque básico
- Eliminación del código no utilizado
- Una mejor asignación de los registros

Ejemplo: Optimización Global

```
IF A(1)<0 & J>0 THEN L:=1 ELSE L:=2;
J:=2;
FOR K:=1 STEP 2 UNTIL J DO
BEGIN
    A(K+1):=A(K)+A(K+1);
    I:=J;
    L:=K*I;
END
```



Ejemplo de Optimización Global
Eliminación de Variables Innecesarias



Se puede eliminar L e I

Las A(K),A(K+1) no se eliminan pues no se han podido considerar en el cálculo de variables vivas

Sección 1.4: De mirilla

La optimización de mirilla trata de estructurar de manera eficiente el flujo del programa, sobre todo en instrucciones de bifurcación como son las decisiones, ciclos y saltos de rutinas.

La idea es tener los saltos lo más cerca de las llamadas, siendo el salto lo más pequeño posible. Este tipo de optimización es aplicable tanto en el código intermedio como en el código objeto, ya que constituye una nueva fase aislada.

La idea básica de este tipo de optimización consiste en:

- Se recorre el código buscando combinaciones de instrucciones que puedan ser reemplazadas por otras equivalentes más eficientes
- Se utiliza una ventana de n instrucciones y un conjunto de patrones de transformaciones (patrón, secuencias reemplazan)
- Si las instrucciones de la ventana encajan con algún patrón se reemplazan por lo secuencia de reemplazamiento asociada
- Las nuevas instrucciones son reconsideradas para las futuras optimizaciones

  Ejemplo: Eliminación de cargas innecesarias
  MOV Ri, X
  MOV X, Rj MOV Ri, Rj

Algunas características de este tipo de optimización son:

- Tiene los saltos lo más cerca de las llamadas, siendo el salto lo más pequeño posible
- Se recorre el código buscando combinaciones de instrucciones que puedan ser reemplazadas. Por otras equivalentes más eficientes
- Se utiliza en: C#, Java, Javacc, Python, Haskell, Ensamblador, Condiciones if, Condiciones while, case

Capítulo 2: Costos

Los costos son el factor más importante a tomar en cuenta a la hora de optimizar ya que en ocasiones la mejora obtenida puede verse no reflejada en el programa final, pero si ser perjudicial para el equipo de desarrollo.

La optimización de una pequeña mejora tal vez tenga una pequeña ganancia en tiempo o en espacio pero sale muy costosa en tiempo en generarla. Pero en cambio sí es optimización se hace por ejemplo en un ciclo, la mejora obtenida puede ser N veces mayor, por lo cual el costo se minimiza y es benéfico la mejora

Ejemplo: for (int i=0; i<10000; i++); si la ganancia es de 30 ms 300 s

Sección 2.1: Costo de ejecución (memoria, registros, pilas)

Los costos de ejecución son aquellos que vienen implícitos al ejecutar el programa. En algunos programas se tiene un mínimo para ejecutar el programa, por lo que el espacio y la velocidad del microprocesador son elementos que se deben optimizar para tener un mercado potencial más amplio.

Otro tipo de aplicaciones que deben optimizarse son las aplicaciones para dispositivos móviles. Los dispositivos móviles tienen recursos más limitados que un dispositivo de cómputo convencional razón por la cual, el mejor uso de memoria y otros recursos de hardware tiene mayor rendimiento.

En algunos casos es preferible tener la lógica del negocio más fuerte en otros dispositivos y hacer uso de arquitecturas descentralizadas como Cliente/Servidor o P2P.

Las aplicaciones multimedia como los videojuegos tienen un costo de ejecución alto por lo cual la optimización de su desempeño es crítica. La gran mayoría de las veces requieren de procesadores rápidos, tarjetas de videos potentes y de mucha memoria

Memoria

La memoria es uno de los recursos más importantes de la computadora y, en consecuencia, la parte del sistema operativo responsable de tratar con este recurso, el gestor de memoria, es un componente básico del mismo.

El gestor de memoria del sistema operativo debe hacer de puente entre los requisitos de las aplicaciones y los mecanismos que proporciona el hardware de gestión de memoria.

Algunos lenguajes de programación se tienen un mínimo para ejecutar el programa, por lo que el espacio y la velocidad de los microprocesadores son elementos que se deben optimizar para tener un mercado potencial más amplio.

Registros

Algunos lenguajes de programación utilizan la pila para almacenar datos que son locales a espacio para los datos locales se asigna a los temas de la pila cuando el procedimiento se introduce, y son borradas cuando el procedimiento termina.

Los registros del procesador se emplean para controlar instrucciones en ejecución, manejar direccionamiento de memoria y proporcionar capacidad aritmética. Los registros son espacios físicos dentro del microprocesador con capacidad de 4 bits hasta 64 bits dependiendo del microprocesador que se emplee.

Los registros son direccionales por medio de una viñeta, que es una dirección de memoria. Los bits, por conveniencia, se enumeran de derecha a izquierda. Estos registros están divididos en seis grupos los cuales tienen un fin específico:

Registros de segmento

- Registros de apuntador de instrucciones
- Registros apuntadores
- Registros de propósito general
- Registros índices
- Registros de banderas

Memoria Estática

La forma más sencilla de almacenar el contenido de una variable en memoria, en tiempo de ejecución, es hacerlo en la memoria estática. Así, el almacenamiento de dichas variables será permanente (durante la ejecución del programa). Por ello, resulta obvio que los datos etiquetados como constantes y las variables globales de un programa tengan asignada la memoria necesaria durante toda la ejecución del programa. Sin embargo, no todas las variables pueden almacenarse estáticamente.

Para que una variable pueda ser almacenada en memoria estática, es necesario conocer su tamaño (número de bytes necesario para su almacenamiento) en tiempo de compilación. Como consecuencia, aunque una variable (u objeto) sea de ámbito global, no podrán ocupar almacenamiento estático.

Pilas

La aparición de lenguajes con estructura de bloque trajo consigo la necesidad de técnicas de alojamiento en memoria más flexibles, que pudieran adaptarse a las demandas de memoria durante la ejecución del programa.

En general los compiladores, la asignación de memoria de variables locales se hace de una forma más flexible, atendiendo al hecho de que solamente necesitan memoria asignada desde el momento que comienza la ejecución de la función hasta el momento en que esta finaliza.

Así, cada vez que comienza la ejecución de un procedimiento (o función) se crea un registro de activación para contener los objetos necesarios para su ejecución, eliminándolo una vez terminada esta.

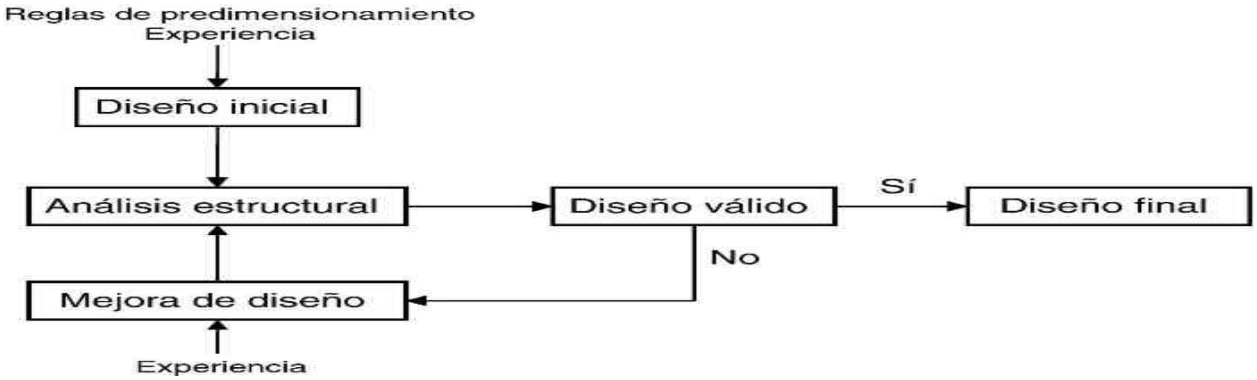
Dado que durante la ejecución, un programa es habitual que unos procedimientos llamen a otros y estos a otros, sucesivamente, se crea cadena jerárquica de llamadas a procedimiento.

Dado que la cadena de llamadas está organizada jerárquicamente, los distintos registros de activación asociados a cada procedimiento (o función) se colocaran en una pila en la que entraran cuando comience la ejecución del procedimiento y saldrán al terminar el mismo.

Sección 2.2: Criterios para mejorar el código

La mejor manera de optimizar el código es hacer ver a los programadores que optimicen su código desde un inicio, el problema radica en que el costo podría ser muy grande ya que tendría que codificar más y/o hacer su código más legible.

Los criterios de optimización siempre están definidores por el compilador.

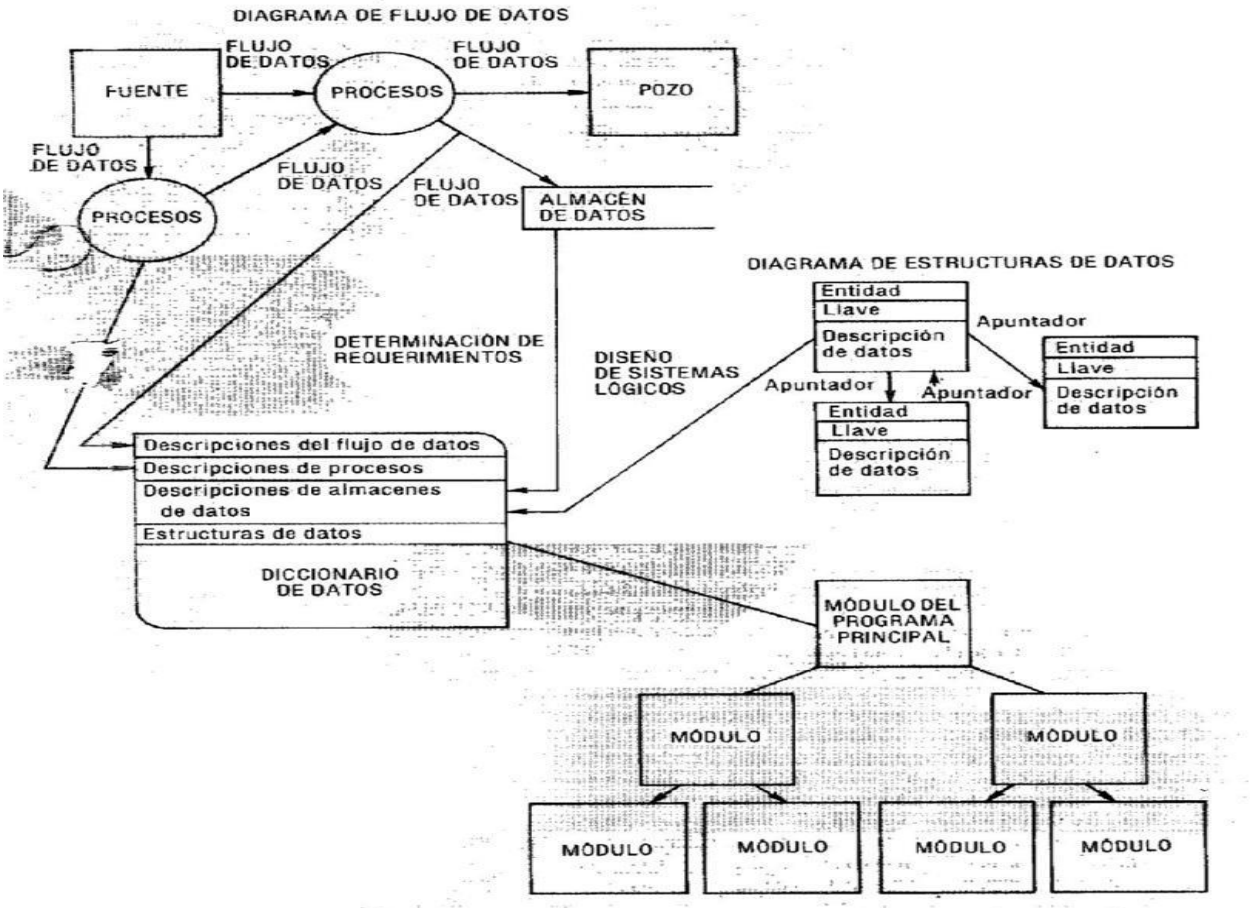


Muchos de estos criterios pueden modificarse con directivas del compilador desde el código o de manera externa
Este proceso lo realizan algunas herramientas del sistema como los ofuscadores para código civil y código para dispositivos móviles

Sección 2.3: Herramientas para el análisis del flujo de datos

El método de desarrollo por análisis estructurado a menudo se interpreta como útil sólo para la parte de análisis de sistemas de la actividad de desarrollo. Algunos analistas lo emplean sólo de esta manera pero su uso no se limita necesariamente a esta fase.

Existen algunas herramientas que permiten el análisis de los flujos de datos, entre ellas tenemos los depuradores y desambladores. La optimización al igual que la programación es un arte y no se ha podido sistematizar del todo.



Algunas ventajas del análisis de flujo de datos son:

1. Los usuarios y otras personas de la empresa que forman parte del proceso bajo estudio comprenden con facilidad anotaciones sencillas. Por consiguiente, los analistas pueden trabajar con los usuarios y lograr que participen en el estudio de los diagramas de flujo de datos.
2. Los usuarios pueden hacer sugerencias para modificar los diagramas con la finalidad de describir la actividad con mayor exactitud. Asimismo pueden examinar las gráficas y reconocer con rapidez problemas
3. El análisis de flujo de datos permite a los analistas aislar áreas de interés en la organización y estudiarlas al examinar los datos que entran en el proceso
4. A medida que los analistas reúnen hechos y detalles, comprenden mejor el proceso

Conclusiones

Como pudimos observar en este resumen, pudimos conocer los tipos de optimizaciones para nuestro código de programación, algo importante a resaltar aquí es que nosotros debemos tomar en cuenta la información de cada uno, ya que tanto hay una optimización que es rápida pero carece de encontrar y arreglar errores, como la de tardarse en compilar el código pero es la más eficiente al de momento de la compilación.

Otra parte importante que debemos tomar en cuenta serían los costos de ejecución para nuestra maquina o incluso nuestro mismo código, ya que debemos de tomar en cuenta varios factores para que, tanto nuestra maquina funcione en condiciones óptimas como la de que nuestro código sea compilado de manera más rápida y sin necesidad de utilizar tantos recursos que nos podremos ahorrar si podemos optimizar el mismo código.

También aprendí que para el análisis del flujo de datos existen herramientas que sirven más que nada para tener una idea más clara y objetiva de cómo queremos que nuestro código funcione de manera más óptima sin necesidad de utilizar memoria que solo será desperdiciada. Estas herramientas las tomo muy en cuenta como si fuera un diagrama UML o un algoritmo, que en vez de ser puro código escrito, es más de utilizar figuras con un uso específico y que ya tienen por predeterminado una función.

Conceptos

Desensamblador: Es un programa de computador que traduce el lenguaje de maquina a lenguaje ensamblador

Código: Conjunto de líneas de texto con los pasos que debe seguir la computadora para ejecutar dicho programa

Flujo de datos: Se utiliza para hacer varias cosas entre ellas trabajos y tareas. Es una representación gráfica del flujo de datos a través de un sistema de información

Análisis: Etapa del ciclo de vida de un sistema informático, esta etapa los analistas se encargan de analizar los requisitos del sistema

Memoria: Dispositivo que retiene, memoriza o almacena datos informáticos durante algún periodo de tiempo

Analizador lexicográfico: Es la primera fase de un compilador consistente en un programa que recibe como entrada el código fuente de otro programa y produce una salida compuesta de tokens

Generador de código: Es una de las fases mediante el cual un compilador convierte un programa sintácticamente correcto en una serie de instrucciones a ser interpretadas por una maquina

Lenguaje de programación: Lenguaje formal diseñado para realizar procesos que pueden ser llevados a cabo por maquinas como las computadoras

Compilación: Es el proceso por el cual se traducen las instrucciones escritas en un determinado lenguaje de programación a lenguaje máquina.

Sockets: Designa un concepto abstracto por el cual dos programas pueden intercambiar cualquier flujo de datos, generalmente de manera fiable y ordenada

Método de puntos: Consiste en asignar una cantidad de puntos a una aplicación informática según la complejidad de los datos que maneja y de los procesos que realiza sobre ellos

Bifurcación: Cuando se aplica en el contexto de un lenguaje de programación o un sistema operativo, hace referencia a la creación de una copia de sí mismo por parte de un programa, que entonces actúa como un proceso hijo del proceso originario, ahora llamado “padre”

Código objeto: Código que resulta de la compilación del código fuente. Puede ser lenguaje maquina o bytecode y pueden distribuirse en varios archivos que corresponden a cada código fuente compilado

Microprocesador: Es el circuito integrado central más complejo de un sistema informático

P2P: Es una red de ordenadores en la que todos o algunos aspectos funcionan sin clientes ni servidores fijos, sino una serie de nodos que se comportan como iguales entre si

Bits: Es un digito del sistema de numeración binario. La capacidad de almacenamiento de una memoria digital también se mide en bits

Bibliografía

Beck. Software de Sistemas, Introducción a la programación de Sistemas. Addison-Wesley Iberoamericana.

Guerra Crespo. Héctor. Compiladores. Ed. Tecnológica didáctica.

Vázquez Gaudioso. Elena, García Saiz. Tomas. Introducción a la teoría de autómatas, gramática y lenguajes. Ed. Universitaria Ramón Areces

Meloni. Brenda, Giro. Juan, Vázquez. Juan, Constable. Leticia. Lenguajes formales y teoría de autómatas. Ed. Alfaomega

Millan Borrajo. Daniel, Fernández Martínez. Paloma, Isasi Viñuela. Pedro. Lenguajes, Gramáticas y Autómatas: Un enfoque práctico. Ed. Addison Wesley

Reporte

Optimización: En este tema se habló sobre el significado general de optimización y como dicha fase se aplica dentro del código intermedio, aplicando el método de revisar el código varias veces del cual se aplica desde la parte del análisis más abstracto hasta el menos abstracto. También tomo en cuenta las ventajas que tiene esta fase si queremos aplicarlo en nuestro código.

Tipos de optimización: Este capítulo solo te da una idea de la variedad de tipos de optimizaciones que podemos aplicar en nuestro código, pero debemos tener en cuenta que lenguaje de programación lleguemos a utilizar y lo más importante de esto, es que gracias a los diferentes tipos de optimización, tienen varias funciones al momento de compilar dicho código.

Locales: Este tipo de optimización funciona principalmente en la sección de procedimientos y clases del código, debido a que su característica principal es que solo sirve cuando un bloque de sección o de programa es crítico e incluso existen varios tipos de optimizaciones locales pero más que nada sirve para que el programador utilice cálculos entre constantes representados explícitamente sin introducir ineficiencias.

Ciclos: Este tipo de optimización es la más fácil de entender debido a que su nombre lo explica todo, debido a que realiza acciones o procesos de manera repetitiva, logrando tratar de encontrar elementos que no deben repetirse en un ciclo, aunque debemos de tomar muy en cuenta que si dicho proceso estaba mal realizado el problema se hará N cantidad de veces, logrando así un bucle infinito.

Globales: Esta optimización es aplicada a todo el código (tanto código objeto como código intermedio), aunque su compilación es la más lenta de todas, mejora el desempeño general de todo el programa sin necesidad de utilizar demasiada memoria. Esta optimización debe depender demasiado de la arquitectura de la maquina por la cual se está trabajando, entre mejor maquina tengas, tus variables globales agilizan todos los procesos pero con la desventaja de consumir más memoria.

De mirilla: Este tipo de optimización es el más eficiente en el flujo del programa al momento de estructurarlo debido a que las instrucciones de bifurcación están más tomadas en cuenta como una toma de decisiones o saltos de rutina. Esta fase es muy aplicada tanto en el código intermedio como en el código objeto, dando así una nueva fase aislada.

Costos: Es el factor más importante a tomar en cuenta si queremos optimizar nuestro código ya que en la mayoría de las ocasiones puede ser perjudicial para la máquina y dando como consecuencia el no poder verse reflejada en el programa final. La única

ventaja de esto sería el obtener una pequeña ganancia en tiempo o espacio pero perjudicando el tiempo por el cual se llegue a generar.

Costo de ejecución (memoria, registros, pilas): En este tipo de costos es donde más se toma en cuenta para ejecutar nuestro programa, tomando muy en cuenta el microprocesador en cuestiones de velocidad, debido a que podemos optimizar un mercado más amplio (tales son los casos en los dispositivos móviles o las aplicaciones multimedia).

Criterios para mejorar el código: Esta sección explicaba más que nada sobre como los programadores pueden optimizar nuestro código desde un inicio y la mayoría de los criterios pueden modificarse con directivas al compilador desde el código. Aunque el único inconveniente seria el que llegara a radicar los costos grandes para lograr que nuestro código sea más legible y tenga una mejora de codificación muy significativa.

Herramientas para el análisis del flujo de datos: Este tema habla principalmente de las herramientas para su análisis del programa, esta sección la entendí mas fácil como si se hablara del diagrama UML, que por medio de formas y/o dibujos que ya tiene definidas sus procesos sin necesidad de tener demasiado código escrito, entre ellas también incluye los depuradores y desambladores, lo malo de todo esto es que es el único método por el cual no se puede sistematizar de todo.