Programación con Asignaciones

- **WHILE** x ≠ A [i] **DO**
  i := i -1
  END
- se manifiestan tres conceptos:
  - **asignaciones**: las variables **i** y **x** denotan localidades de la máquina en la cual esta implementado el lenguaje
  - **estructuras de datos asignables**:una estructura es asignable si cuenta con componentes cuyos valores pueden cambiarse por medio de asignaciones, en este caso **A[i]**
  - **enunciados de flujo de control**: el flujo de control es un programa, especificado por enunciados. Las palabras claves **WHILE, DO** y **END**; constituyen el enunciado

- MODULA - 2 y C son ejemplos de *programación imperativa*; que reconstruyen la *máquina* en la que están implementados, para hacerla más *adecuada* a la programación
- las *máquinas* determinan que lenguajes imperativos manejan
- la *adecuación*, o facilidad de programación se refiere al estilo de programación que el lenguaje puede manejar

- Las máquinas determinan el siguiente principio para el diseño de lenguajes:

  - **modelo de máquina**: todo lenguaje debe permitir una asignación orientada a la máquina en la que esta implementado, para tener un uso directo y eficiente
  - **el estilo**; programación estructurada: la estructura del texto del programa debe auxiliarnos para entender la función del programa

Evolución de los lenguajes imperativos ...

- El desarrollo de los lenguajes imperativos está marcado por la aparición ocasional de un sucesor evolucionado, bajo un nombre nuevo

De Algol 60 a Pascal y de Pascal a Modula - 2

- El diseño en los lenguajes en el decenio de 1960 fue dominado por los intentos para mejorar Algol 60 (mejor que sus antecesores y cercano a sus sucesores).
- Inició varias tradiciones:
  - uso de la notación BNF para especificar la sintaxis y su empleo en el manual de referencia.

- Limitaciones de Algol 60:
  - los arreglos eran la única estructuras de datos y
  - constructores de enunciados eran muy complejos

- La secuencia de lenguajes diseñados por Niklaus Wirth, incluyendo Pascal y Modula - 2 , muestra la evolución de los lenguajes imperativos a partir de Algol 60

Algol W. Wirth y Hoare [1966]:

- una gran parte es como Algol 60, se mejoran los recursos para la estructuración de datos

- los cambios a los recursos relacionados con el control de secuencias han ido en direcciónde:
  - simplificación
  - clarificación

Pascal según Wirth [1971]

- Algol W, es l antecesor directo de Pascal
- constituyó la fuente de muchas características como:
  - los enunciados como: while, case y estructuras de registros

Modula - 2 Wirth [1983]

- Incluye todos los aspectos de Pascal y los amplía
  - concepto de módulo
  - cuanta con una sintaxis más sistemática que facilita el proceso de aprendizaje
  - cada estructura que comienza con una palabra clave, termina con una palabra clave *(esta delimitada apropiadamente)*

Oberón: Wirth [1988] ...

- Evolucionó a partir de Modula con algunas adiciones y varias omisiones. Basándose en la evolución más que en el revolución, permanecimos en la tradición del largo desarrollo que partió de Algol hacia Pascal, hacia Modula -2, y por último a Oberón

De Algol 60 a BCPL y de BCPL a C ...

- Los lenguajes del árbol familiar de C fueron trabajo de muchas personas:
  - CPL (Combined Programming Language), Strachey [1966]
    - el lenguaje se conservó en el laboratoriopara estudiar conceptos; nunca se impleemntó totalemente.
    - Uno de los objetivos era hacerlo una práctica de una teoría coherente y lógica sobre lenguajes de programación

- BCPL (Basic CPL); Richards [1969] lo desarrolló, como herramienta para la construcción de compliadores:
  - adoptó mucho de la riqueza sintáctica CPL
  - luchó por conservar el mismo nivel de elegancia lingüística, sin embargo ...
  - para lograr la eficiencia necesaria en la programación de sistemas, su escala y complejidad están más allá de las de CPL

C, Dennis Ritchie [1972] ...

- Fue creado como lenguaje de implementación para programas asociados con el sistema operativo UNIX
  - en 1973 el sistema operativo UNIX se re-escribe en C
  - C proporciona un buen conjunto de operadores, una sintaxis concisa y un acceso eficiente a la máquina
  - es un lenguaje de propósito general y está disponible en un gama extensa de compiladores

C++ Stroustrup [1986] ...

- Además de las facilidades proporcionadas por C
  - proporciona recursos flexibles y eficientes para definir tipos nuevos
  - acepta los programas en C con algunos pequeños cambios
  - los nuevos tipos se definen usando clases; utilizadas originalmente en Simula 67

- La diferencia entre el C de hoy y el de 1972 es:
  - la verificación de tipos más estricta
  - el sistema de tipos se amplió en 1977, para mejorar las transportabilidad de los programas de C

- un proyecto para trasladar UNIX de una máquina a otra reveló un amplio espectro de violaciones en la verificación de tipos dentro de un programa que podía ejecutarse en una máquina y en otra no
- entre las violaciones más terribles estaba: la confusión entre apuntadores y enteros
- C++ tiene un *sistema de tipos estricto*

Formato de Impresión para los programas

- Pascal
  - **while** x ≠ A[i] **do**
    i := i -1
- Modula - 2
  - **WHILE** x ≠ A[i] **DO**
    i := i -1
    END
- C
  - while (x ! = A[i])
    i = i - 1;

Efecto de una asignación

- Una asignación cambia el estado de la máquina; donde el estado corresponde comparativamente a una fotografía de la memoria de la máquina.

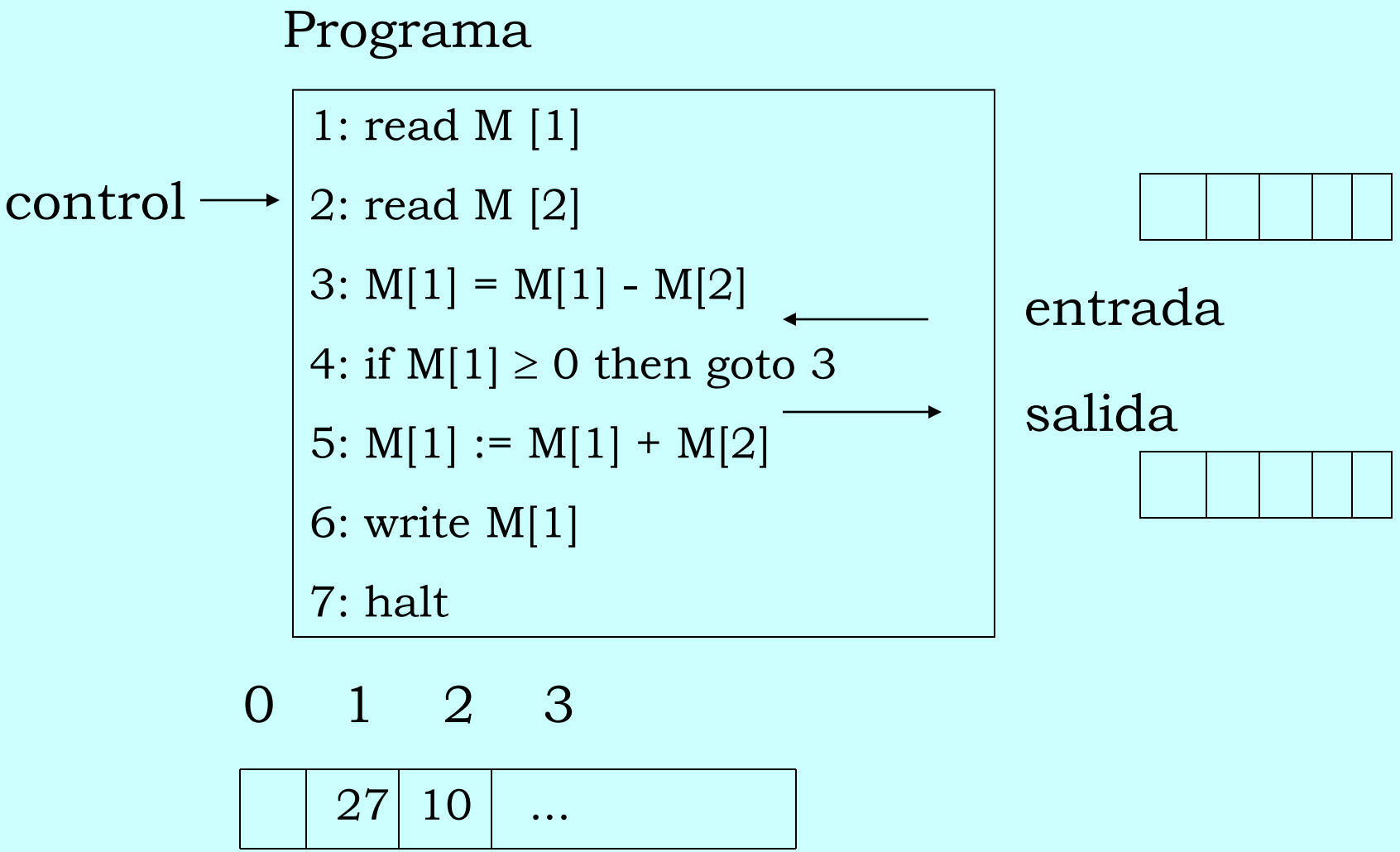
- Los conceptos de: estado, localidad y valor provienen del modelo de computadora para los lenguajes imperativos

Máquinas de acceso aleatorio

- Una máquina de acceso aleatorio (RAM) tiene:
  - una memoria
  - un programa
  - un archivo de entrada y
  - un archivo de salida

La memoria consiste ...

- secuencia de localidades 0, 1, ...
- capaz de almacenar un entero a la vez ...
- la dirección de máquina es el número de una localidad en memoria
- el entero almacenado en una localidad se identifica como el contenido de una localidad



Máquina de acceso aleatorio RAM

El programa

- consiste en una secuencia de instrucciones ...
- El conjunto de la siguiente figura tiene:
  - **instrucciones para asignaciones**
    - <expresión> $_1$:= <expresión >$_2$
    - localidad de memoria y valor
  - **entrada y salida :** secuencia de valores tomados de uno en uno por instrucciones de la forma **read M[l] / Write M[j]**
  - **flujo de control**

Conjunto de instrucciones para la máquina de acceso aleatorio ...

- **Asignaciones:**
  - M[l] := n
  - M[l] := M[j] + M [k]
  - M[l] := M[j] + M [k]
  - M[l] := M[M[j]] {dirección indirecta}
  - M[M[j]] := M[k]

- Entrada/Salida
  - **read** M[l]
  - **write** M[j]

- Flujo de control
  - **if** M[j] >= 0 **then goto** i
  - **halt**

- Valores *i* y valores *d*
  - *i* para localidad = lado izquierdo
  - *d* para valor = lado derecho

Seguimiento dinámico del control a través del programa

- un cálculo dinámico puede visualizarse como un hilo dejado por el flujo de control a través del texto estático del programa

- el efecto de un seguimiento del cálculo en una RAM se describirá tomando instantáneas llamadas estados

El estado tiene tres partes:

- una correspondencia entre localidades y sus contenidos
- el resto de la secuencia de entrada y
- la secuencia de salida producida hasta el momento

- es importante mencionar que las instrucciones de asignación y de entrada/salida cambian el estado, sin interferir en el flujo normal del control de una instrucción a la siguiente.

- las instrucciones de flujo de control conducen al seguimiento sin cambiar el estado de la RAM

Puntos a aclarar ...

- Los lenguajes imperativos se conocen como Von Neumann, sin embargo la máquina RAM es más adecuada, debido a que los programas RAM son estáticos al igual que los programas imperativos...

- La máquina de Von Neumann modifica su programa durante la ejecución para superar su carencia de direccionamiento indirecto {ejemplo de la página 73}

Programación estructurada

- El acercamiento sistemático al diseño de programas surge de ejemplos como el siguiente ...

- Ejemplo Bentley [1986], pidió a más de cien programadores profesionales convertir la siguiente descripción de búsqueda binaria ‘en un programa escrito en el lenguaje de su prefrencia; un pseudocódigo de alto nivel también sería aceptable

- Nos informa, estoy sorprendido; teniendo tiempo suficiente sólo cerca del 10% de los programadores fueron capaces de realizar bien este pequeño programa.

Descripción de la búsqueda binaria

- Determinar si el arreglo ordenado x[1..N], contiene el elemento T. La búsqueda binaria resuelve el problema, determinando el intervalo del arreglo en el cual T debe encontrarse, en caso de hallarse dentro del arreglo. Inicialmente este intervalo es todo el arreglo, el intervalo se reduce comparando su elemnto medio con T y descartando una de las mitades. El proceso continúa hasta encontrar T en el arreglo o cuando el intervalo donde debiera hallarse es el intervalo nulo

- la programación estructurada es un método para desarrollar programas correctos y entendibles

- surgió de un enérgico debate sobre el mérito de los constructores para el flujo de control, iniciado por Dijktra en un artículo titulado ´Go To statement considered harmful’

Ideas que combina la programación estructurada

- **Flujo de control estructurado:** un programa es estructurada si si el flujo de control es evidente a partir de *la estructura sintáctica del texto del programa*.

- **Invariantes:** es *una afirmación* en el punto *p* y que se sostiene cada vez que el control alcanza el *punto p*

- **Invariantes:** es una afirmación que puede ser falsa o verdadera acerca del estado de un cálculo. Ejem: $x \geq y$ que relaciona los valores de x y y

- las invariantes se utilizan para la redacción de programas estructurados

Enunciados atómicos

- Existen varios tipos de enunciados:
  - enunciado de asignación
    - <expresión> := <expresión>
  - invocaciones de procedimientos
    - <nombre del procedimiento> (<parámetros reales>)
  - termina un programa y devuelve <expresión> como resultado
    - **return** <expresión>

- Las asignaciones suponen la existencia de una máquina en la que se implementa el lenguaje, capaz de almacenar varios tipos de valores básicos: *booleanos*, *caracteres*, *enteros y reales* y como estructura de datos *los arreglos*

- Flujo de control estructurado

  - **Composición:** Si S_1, S_2, ...S_k son enunciados, $k \geq 0$, entonces su composición es una lista de enunciados que se escribe así: S_1; S_2; ...; S_k

  - **Condicional:** Si E es una expresión y $\mathbf{LE_1}$ y $\mathbf{LE_2}$ son listas de enunciados, entonces un enunciado condicional formado por esos elementos sería:

    if E **then** LE_1 **else** LE_2 **end / if** E **then** LE_1 **end**

– **Ciclo infinito:** si LE es una lista de enunciados, entonces una interacción es es:

loop LE **end**

la ejecución de un enunciado **exit** envía el control fuera del **loop** y al enunciado que se haya inmediatamente después del ciclo

– **Ciclo while:**

while E **do** LE **end**

la expresión E se evalúa alternativamente y LE se ejecuta mientras E sea *verdadera*. En el instante que sea *falsa* el control se dirige al enunciado que sigue al ciclo **while**

Los invariantes relacionan programas y ejecución

- Una de las dificultades para escribir código correcto es que la correctez es una propiedad que no pertenece al texto fuente estático, sino a su ejecución dinámica; *cuando el programa se ejecuta ambas cosas se toman en cuenta*.

- los invariantes pueden ayudarnos a relacionar ambas cuestiones.

- están atados a un punto del programa y nos indican una propiedad de sus cálculos, en forma tal que relacionan el texto estático del programa y el seguimiento dinámico de sus cálculos.

- **while** x ≥ y **do**

  x := x - y;

  end

- Cada vez que se alcanza la asignación es porque se cumplió la condición booleana

- El mismo ejemplo con la invariante es:

- **while** $x \geq y$ **do**

  {si llegamos aquí, $x \geq y$}

  x := x - y;

  end

- El lugar preferido para colocar una invariante de un ciclo **while** es el punto antes de probar E y se le conoce como invariente del ciclo

  while {invariante de ciclo} E **do** LE **end**

- Otros tipos de invariantes son:
  - **precondición;** se coloca antes del enunciado
  - **postcondición;** se coloca después del enunciado

- **loop**

  $\{x \geq 0$ y $y > 0\}$
  while x ≥ y **do**

  x := x - y;

  end

- la primera vez que el control entra al ciclo; se supone que el *invariante* es verdadero

- la condición $x \geq y$ entre while y do asegura que x sea mayor que y antes de la asignación $x := x - y$, despúes de esta el nuevo valor de x debe satisfacer $x \geq 0$ y de esta manera el *invariante* debe sostenerse
- {78}

Tipos de datos en MODULA -2

- En los lenguajes imperativos, los tipos se usan para:
  - verificar errores y
  - establecer la disposición de los datos en la máquina en la que se implanta el lenguaje
- hay que recordar que cada tipo tiene asociado un conjunto de operaciones

- las expresiones de tipo describen la estructura de un tipo de datos
- las estructuras simples son nombres de tipo como: INTEGER
  - ARRAY [1..99] OF INTEGER

- En MODULA se permite aplicar constructores de tipos en cualquier orden con el fin de *crear estructuras de datos jerárquicas*

  – arreglos de arreglos, arreglos de apuntadores a registros

    - **POINTER TO RECORD**
    - re, im : **REAL**
    - **END**

- *ExpresionTipo* ::= *TipoSimple*
  {*NombreTipo*
  |**ARRAY** *TipoSimple* **OF** *ExpresionTipo*
  | **RECORD** {*Nombre*{‘,’*Nombre*}’:’*ExpresionTipo*’;’}**END**
  |**POINTER TO** *ExpresionTipo*
  |**SET OF** *TipoSimple*

  TipoSimple ::= *TipoBasica*/*Enumeracion*/*Subintervalo*

  TipoBasico ::= **BOOLEAN** |**CHAR** |**CARDINAL** |**INTEGER** |**REAL**

  Enumeracion ::= ‘(’*Nombre*{‘,’*Nombre*}’)’

  Subintervalo::= [*NombreTipo*]’[‘*ExpresionConstante*’..’*ExpresionConstante*’]’

Ejemplo de un programa en MODULA - 2

- {de mi stock}