Clase 4 – Unidades

.Los lenguajes de programacion permiten que un programa este compuesto por UNIDADES. En general se las llama rutinas(procedimientos/funciones).

Caracteristicas sintacticas, semanticas de las rutinas y mecanismos de control del flujo de ejecucion entre rutinas con ligaduras involucradas:

NOMBRE: Se usa para invocar a la rutina. El nombre se introduce en su declaracion y se usa para invocarlas.

ALCANCE: Es el rango de instrucciones donde se conoce su nombre. El alcance se extiende desde el punto de su declaracion hasta algun constructor de cierre. Según el lenguaje puede ser estatico o dinamico. La invocacion puede estar solo dentro del alcance de la rutina.

Algunos lenguajes distinguen a la declaracion como el encabezado (por ej: public void sumar()) y la definicion como el todo. Si esto sucede permite manejar un esquema de rutinas mutuamente recursivas. Esto es dado a que se puede declarar la funcion A, luego definir la funcion B que llama a la funcion A y debajo de la definicion de B definir a la funcion A. Aunque la funcion no este definida, al estar declarada la funcion B puede llamar a la A. (IMPORTANTE)

.En algunas implementaciones de C, si no se encuentra la declaracion de la funcion, se asume un prototipo que devuelve enteros.

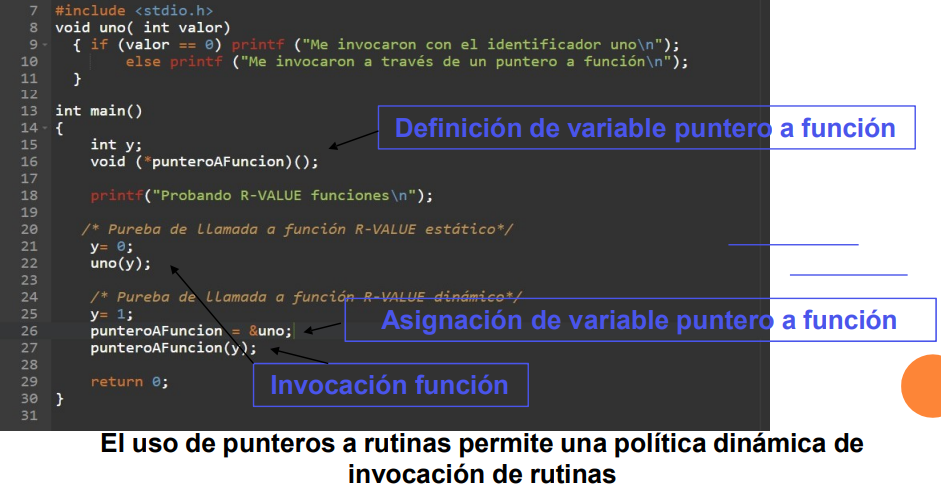
TIPO: El encabezado de la rutina define el tipo de los parametros y el tipo del valor del retorno (si lo hay).

.Signatura: Una rutina fun que tiene como entrada parametros de tipo T1,T2 y devuelve un valor de tipo R, puede especificarse como: fun: T1xT2 -> R

.Un llamado a una rutina es correcto si esta de acuerdo al tipo de la rutina.

.La conformidad requiere la correspondencia de tipos entre parametros formales y reales.

L-VALUE y R-VALUE: El L es el lugar de memoria en el que se almacena el cuerpo de la rutina y el R se constituye por la llamada a la rutina que causa la ejecucion de su codigo. Este ultimo puede ser estatico, el caso mas usual o dinamico, usa variables de tipo rutina (se implementan a traves de punteros a rutinas).



Comunicación entre rutinas: Hay dos posibilidades Ambiente no local y parametros.

.Parametros: Permite mayor legibilidad y modificabilidad.

.Parametros formales: los que aparecen en la definicion de la rutina.

.Parametros reales: los que aparecen en la invocacion de la rutina.

Ligadura entre parametros formales y reales:

.Metodo posicional: se ligan uno a uno. Los parametros reales Ri se ligan a los parametros formales Fi por posicion para i de 1 a N. Deben conocerse las posiciones.

.Variante: es una combinacion con valores por defecto. Ej: C++: int distancia(int a = 0, int b = 0)

distancia() -> distancia(0,0) y distancia(10) -> (10,0)

.Metodo por nombre: se ligan por el nombre, deben conocerse los nombres de los formales.

Representacion en ejecucion: la definicion de la rutina especifica un proceso de computo, cuando se invoca una rutina se ejecuta una instancia del proceso con los particulares valores de los parametros.

.Instancia de la unidad: es la representacion de la rutina en ejecucion. Las instrucciones de la unidad se almacenan en la memoria de contenido fijo y los datos locales de la unidad se almacenan en la memoria de contenido cambiante.

PROCESADOR ABSTRACTO:

PA-Utilidad: sirve para comprender que efecto causan las instrucciones del lenguaje al ser ejecutadas.

PA-Simplesem: se conforma por:

Memoria de codigo: C(y) valor almacenado en la y-esima celda de la memoria de codigo. Comienza en 0.

Memoria de datos: D(y) valor almacenado en la y-esima celda de la memoria de datos. Comienza en 0 y representa el L-valor, D(y) o C(y) su R-valor.

IP: puntero a la instrucción que se esta ejecutando. Se inicializa en 0, en cada ejecucion se actualiza cuando se ejecuta cada instrucción. Utiliza las direcciones de C.

En ejecucion: se deben obtener la instrucción actual para ser ejecutada (C[ip]), se incrementa ip y se ejecuta la instrucción actual.

PA-Instrucciones que acceden a D:

SET: setea valores en la memoria de datos.

Set target, source: copia el valor representado por source en la direccion representada por target.

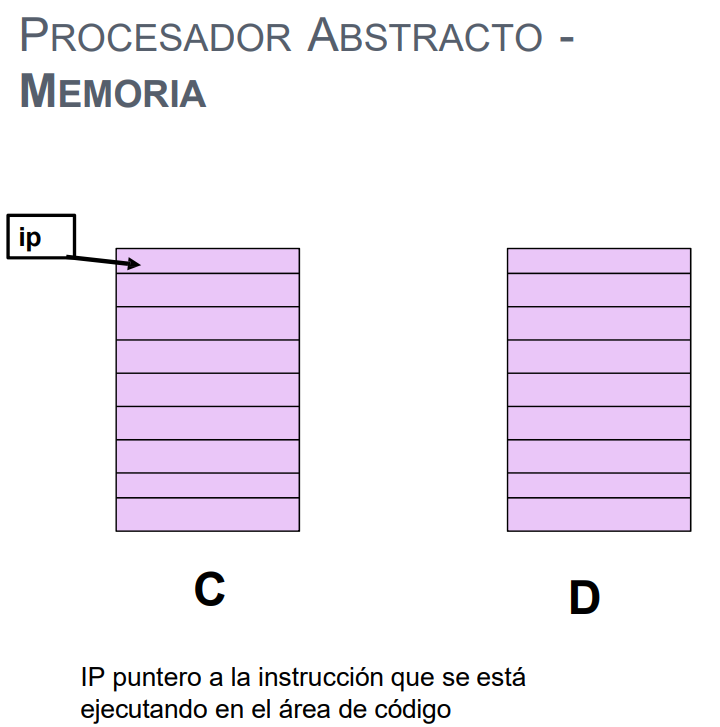
Set 10,D[20]: copia el valor almacenado en la posicion 20 en la posicion 10.

Tambien se puede usar para entrada salida. Ej: set 15, read (el valor leido se almacenara en la direccion 15) y set write, D[50] (se transfiere el valor almacenado en la posicion 50).

JUMP: bifurcacion incondicional a direcciones de C. Ej: jump 47 (la prox instrucción a ejecutarse sera la que este almacenada en la direccion 47 de C).

JUMPT: bifurcacion condicional, bifurca si la expresion se evalua como verdadera. Ej; jumpt 47, D[13] > D[8].

Direccionamiento indirecto: no lo entendi.



Elementos en ejecucion:

.Punto de retorno: es una pieza cambiante de info que debe ser salvada en el registro de activacion de la unidad llamada.

.Ambiente de referencia:

.Ambiente local: usa variables locales, ligadas a los objetos almacenados en su registro de activacion.

.Ambiente no local: usa variables no locales, ligadas a objetos almacenados en los registros de activacion de otras unidades.

Estructura de ejecucion de los lenguajes de programacion:

Estatico (espacio fijo): el espacio necesario para la ejecucion se deduce del codigo, todos los requerimientos de memoria necesarios se conocen antes de la ejecucion, la alocacion puede hacerse estaticamente, no puede haber recursion ya que no pueden mantener diferentes instancias, este es habitual en leguajes estaticos como las versiones originales de FORTRAN y COBOL.

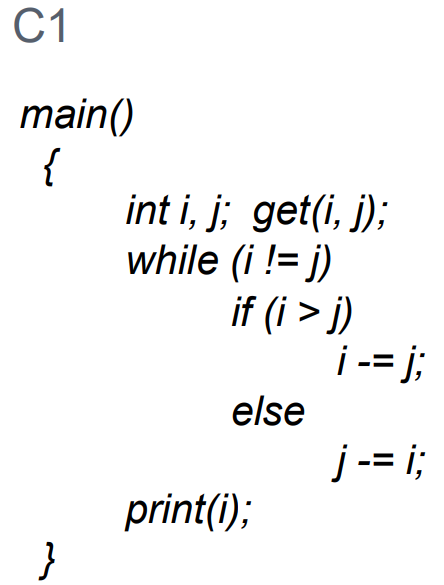
Basado en pila (espacio predecible): el espacio se deduce del codigo (Algol-60). Es para programas mas potentes cuyos requerimientos de memoria no pueden calcularse en traduccion, la memoria a utilizarse es predecible y sigue una disciplina last-in-first-out. Las variables se alocan automaticamente y se desalocan cuando el alcance se termina, se utiliza una estructura de pila para modelizarlo.

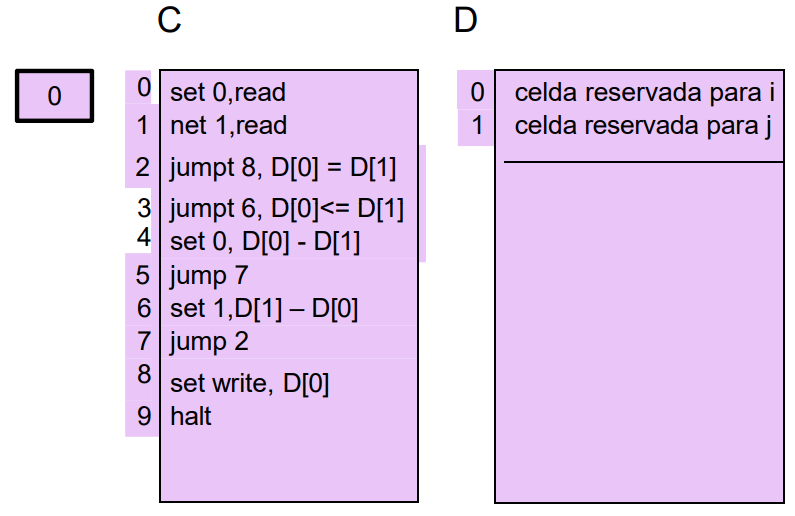
Dinamico (espacio impredecible): lenguajes con impredecible uso de memoria. Los datos son alocados dinamicamente solo cuando se los necesita durante la ejecucion. No pueden modelizarse con una pila, el programador puede crear objetos de dato en cualquiere punto arbitrario durante la ejecucion del programa. Los datos se alocan en la zona de memoria heap.

Esquemas de ejecucion: Casos C1, C2 y C2’: Paginas 34 a 49, leer desde PDF ya que hay mucha explicacion.

.Explicacion de programas y como funcionan teniendo en cuenta el esapcio de codigo y de datos.

C1:

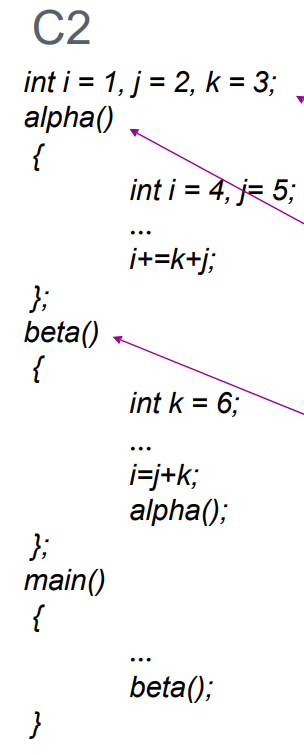


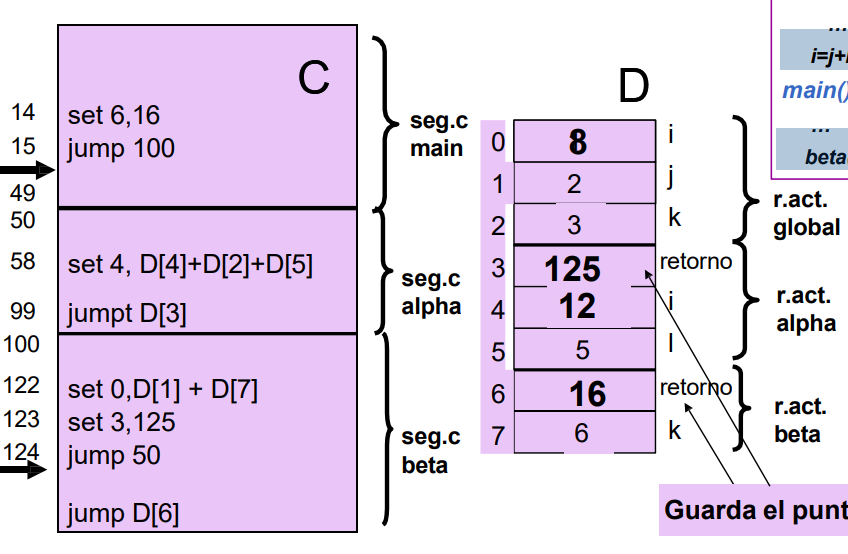


C2 (C1 + Rutinas Simples): El tamaño de cada R.A de cada unidad puede determinarse en compilacion. Todos los R.A pueden alocarse antes de la ejecucion (alocacion estatica), se invoquen o no. Cada variable puede ser ligada a direcciones de memoria D antes de ejecucion.

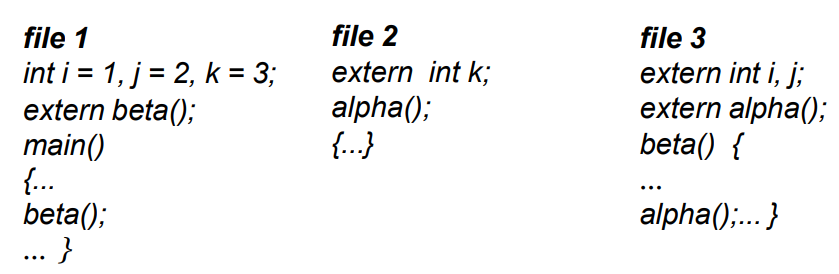
Rutinas internas: no pueden estar anidadas (son disjuntas) y no son recursivas. Ademas manejan datos locales y globales.

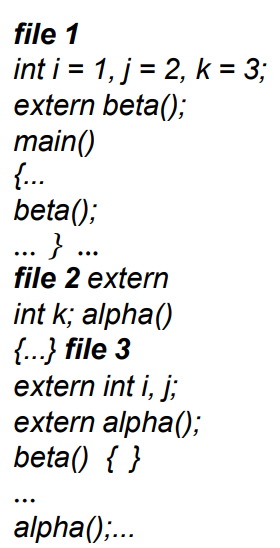
Call-Return: antes de saltar del main a una rutina o de una rutina a otra se guarda el punto de retorno. En el espacio de codigo y de datos “de la zona a la cual se debe retornar”.

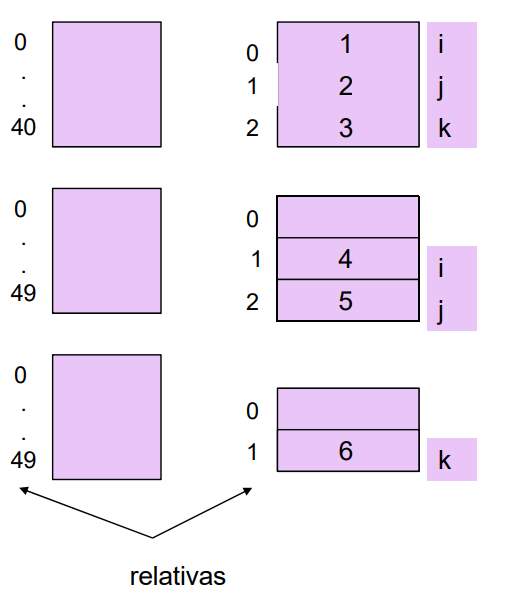




C2’ (C2 pero con rutinas compiladas por separado): En este caso, las unidades del programa se encuentran en archivos separados. Cada archivo es compilado por separado y en orden arbitrario.







En esta caso, como cada unidad es compilada en distinto orden y por separado, entonces el compilador ya no puede: ligar variables locales a direcciones absolutas, tampoco variables globales pueden ligarse con sus desplazamientos en el R.A global y las invocaciones a las rutinas no pueden ligarse con la direccion de inicio de los correspondientes segmentos de codigo.

Surge el Linkeditor: este se encarga de asignar los varios segmentos de codigo a almacenamiento en C, se encarga de asignar los varios registros de activacion dentro de D y completa toda info faltante que el compilador no podia evaluar. C2 y C2’ no difieren semanticamente, una vez que el linkeditor reuna todas las unidades compiladas separadamente.