Notas de curso de Programación y estructuras de datos. Memoria dinámica

Nikos Mylonakis, Fernando Orejas

nicos@lsi.upc.edu

Dept. Llenguatges i Sistemes Informátics Universitat Politécnica de Catalunya

Barcelona

Contenido

- Introducción a los LP
- Memoria dinámica
- Control de datos
- Tipos de datos
- Especificación algebraica
- Implementación de tipos abstractos de datos

Introducción a los LP

- Qué es un lenguaje de programación. Clases de lenguajes
- Criterios de diseño.
- Componentes de un LP.
- Implementación de un LP

- LP. Notación formal que permite escribir todos los programas que pueden ser ejecutados por un computador.
- Los lenguajes que describiremos son de alto nivel
- Diferencia entre algoritmo y programa

Clases de LP

- Imperativos y orientados a objetos (OO)
  - Tiene instrucción de asignación
  - No tienen transparencia referencial
- Declarativos
  - Tienen transparencia referencial
  - Implementación más ineficiente

Criterios de diseño

- Distinción cualitativa entre LP dependiendo del próposito del LP
- Los criterios más importantes son:
  - Eficiencia en la implementación
  - Fiabilidad
  - Usabilidad

Eficiencia

- En los años 50-60 era EL criterio de diseño
- Hoy en día también se valora la facilidad de programar
- Ejemplo1: derivador simbólico en Prolog (25-30 líneas y en C++ (3.000))
- Sólo si la aplicación se ha de utilizar muchas veces al día C++ es más interesante
- Ejemplo2: Java es menos eficiente que C++ pero más portable y seguro

Fiabilidad

- Facilidad del LP para detectar un error lo antes posible.
- La comprobación de tipos en tiempo de compilación ayuda a detectar errores antes de la ejecución.
- Ejemplo 1: i:=i+’a’ i:natural
- Hay comprobaciones que no se pueden realizar en tiempo de compilación
- Ejemplo 2: Rango del índice de los vectores (t[i*2] i:natural t:**tabla** [1..N] **de** entero

Usabilidad

- Hay LP que se diseñan para resolver una cierta clase de problemas
- La evaluación de un LP ha de ser para su dominio de aplicación
- Por tanto un LP puede ser mejor para una clase de problemas y peor para otras

Componentes de un LP. Tipos de datos

- Se ha de determinar sus tipos básicos, constructores de tipos, relación de subtipos
- Se ha de determinar la comprobación de tipos que se hace en tiempo de compilación
- Y la que se hace en tiempo de ejecución (ej: índice de tablas)

Control de secuencia

- No hablaremos en la asignatura.
- Determina cómo se regula el flujo de los programas
- En LP imperativos tenemos composiciones condicionales e iterativas.
- Ejemplos de control más avanzados son la reescritura de términos y el paralelismo

Control de datos (CD) y Unidades de programas (UP)

- CD: Mecanismo que ofrecen los LP para acceder, modificar o mover los datos de un programa
- Ejemplos: Paso de parámetros, ámbito de las variables y parámetros
- UP: Forma de estructurar los programas
- Ejemplos: Clases en LPOO, módulos, funciones

Implementación de un LP

- Compilación: Proceso de traducción de un programa en un LP a código máquina.
- Interpretación: Programa que va ejecutando un programa a medida que lo va leyendo.
- Tradicionalmente se hablaba de compiladores (+ eficientes) e intérpretes (-eficientes)
- Hoy en día no existen LP 100% compilados ni 100% interpretados

- Hoy en día los intérpretes compilan y optimizan el código de los bucles
- Y los compiladores utilizan máquinas virtuales intermedias por ejemplo para E/S mediante el uso de librerías. Por eso es necesario linkar los programas
- Otro motivo que justifica la necesidad de linker es la programación modular

Memoria dinámica

- Gestión de memoria en pila y en heap
- Punteros y operaciones básicas. Ejemplos
- Asignación y liberación de memoria dinámica
- Algoritmos de garbage collection

Gestión de memoria en pila

- Veamos como gestionar la memoria de las variables requeridas en un programa con un conjunto de acciones
- Ejemplo

accion $Principal$	**accion** $A1$	**accion** $A2$
var $x, y, z : \ldots$	**var** $a, b : \ldots$	**var** $c, d : \ldots$
fvar	**fvar**	**fvar**
$\vdots$	$\vdots$	$\vdots$
$A1;$	$A2;$	$A2;$
$\vdots$	$\vdots$	$\vdots$
faccion	**faccion**	**faccion**

Memoria requerida después de ejecutar la acción principal, A1,A2 y una llamada recursiva

x	y	z	a	b	c	d	c	d	

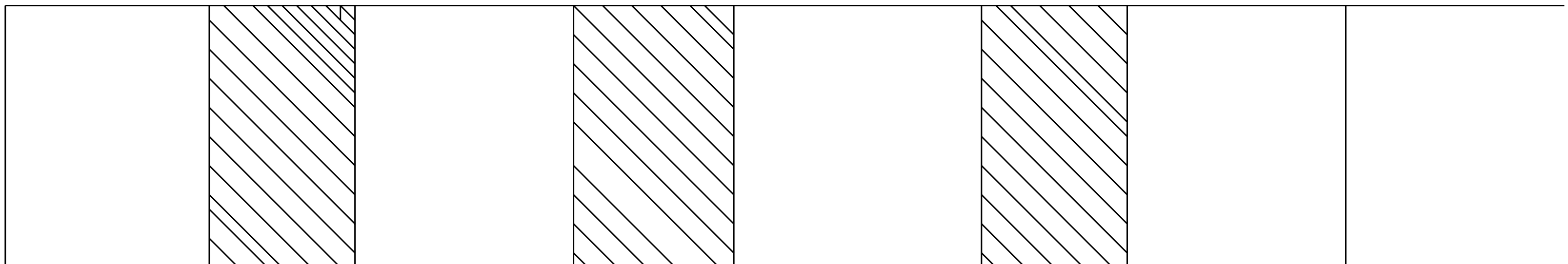
Descripción del proceso de obtención y liberación de espacio y el concepto de bloque de activación.

- La estructura de datos requerida para gestionar la memoria de este tipo de variables es una pila.
- Como ya vimos una pila es una estructura lineal cuyas operaciones principales son las de apilar y desapilar.
- Esta estructura de datos es facilmente implementable mediante una tabla

- En el ejemplo la obtención de memoria se haría con la operación apilar y la liberación de memoria con desapilar.
- Esta gestión de memoria es limitada.
- Para tener estructuras de datos dinámicas mediante el uso de punteros esta gestión de memoria no es suficiente.
- La gestión de memoria para el caso de datos general requiere de la estructura de datos de un heap.

Heap

- Mediante un heap la memoria libre y ocupada en general está entremezclada.
- Cuando se requiere una cierta cantidad de memoria libre se busca en la zona libre un espacio que cubra esa cantidad.
- También es posible liberar memoria que ya no es requerida por el programa.
- Representación gráfica



- La gestión de memoria mediante heap es más costosa
- Para asignar memoria se requiere realizar una búsqueda por las zonas libres
- Aparece el problema de fragmentación. La solución es compactar que agrava la ineficiencia

Punteros o referencias

- Para poder definir estructuras de datos dinámicas los LP ofrecen los punteros o referencias
- Los punteros son constructores de tipos donde se requiere un tipo adicional (generalmente básico o una tupla)
- La sintaxis es $\uparrow T$. Ej: x: $\uparrow$ entero
- Los valores de $\uparrow T$ son direcciones de memoria donde se guardan elementos de tipo T más la dirección distinguida NIL.
- En el ejemplo x denota la dirección de memoria de una variable entera y $x \uparrow$ es una variable entera

Operaciones sobre punteros

- Nosotros asumiremos que el valor inicial de un puntero es la dirección NIL.
- Para asignarle un espacio de memoria del tipo determinado se tiene la operación $reservar(x)$
- Podemos realizar la asignación de punteros (aliasing) pero normalmente no se tienen operaciones aritméticas (en C++ sí que existen)
- Ejemplo

$$\textbf{var } y, x :\uparrow entero \textbf{ fvar}$$

$$reservar(x);$$

$$y := x;$$

- Liberación de la zona de memoria mediante la acción $liberar(x)$.
- El efecto es que x vuelve a tomar el valor NIL y la zona anteriormente asignada a x pasa a la zona libre
- Problema de esta operación con aliasing
- En el ejemplo y pasa a ser referencia colgada (dangling pointer). (En Java no es posible)

$$\mathbf{var}\ y, x :\uparrow entero\ \mathbf{fvar}$$

$$reservar(x); y := x; liberar(x)$$

Algoritmos de asignación y liberación de memoria dinámica

- Presentamos diferentes algoritmos para las acciones de reservar y liberar memoria
- La representación del heap es normalmente mediante una lista encadenada de zonas libres
- Un posible algoritmo para reservar es buscar la primera zona libre con la suficiente memoria requerida

- Problema de fragmentación. Ejemplo: zonas libres de 500, 200 y 300 y se realizan las acciones reservar(200);reservar(150);reservar(250);reservar(250)
- Alternativas: Empezar la búsqueda por la última zona libre usada de forma circular (first fit)
- Buscar la zona de memoria que mejor se ajusta a la zona requerida (best fit)
- Buscar la zona de memoria que peor se ajusta a la zona requerida (worst fit)
- Estudios estadísticos demuestran que el first es en general el mejor

Liberación de memoria

Para liberar una direccion d de longitud N de una zona ocupada podemos:

- Liberar sin actualizar la lista de libres
- Actualizar la lista de libres recorriéndola.
- Resolver problemas de fragmentación en el caso en que la zona liberada es contigua a una zona libre

Recolector de basura (Garbage collector)

- Se entiende por basura la memoria que no está en la zona libre y no es alcanzable por los punteros de un programa.
- Un recolector de basura se encarga de recoger la basura situándola en la zona libre
- Hay dos tipos:
  - Los de **parada/recogida** se clasifican en los que realizan copia y los que realizan marcado y búsqueda (mark/scan).
  - Los **continuos** pueden realizarse mediante contador de referencias o sobre la marcha (on-the-fly)

Parada/recogida Copia

- Divide el heap en dos partes iguales. Una mitad nunca se utiliza porque va alternándose.
- Cuando se requiere memoria se empieza la copia de forma compacta sin copiar la basura
- Así la memoria libre crece pero la mitad copiada queda ahora sin utilizar
- La copia de toda la memoria referenciada es costosa y complicada
- Si hay mucha basura el algoritmo es más eficiente pues lo más costoso es la copia

Parada/recogida Mark/scan

- En este caso se utiliza toda la zona de memoria disponible
- Hay un proceso de marcado de la zona ocupada para diferenciarla de la basura
- Posteriormente se compacta y se desmarca
- El algoritmo es complicado
- Si hay poca basura el algoritmo es más eficiente pues lo más costoso es la compactación

Continuo contador de referencia

- Cada zona de memoria referenciable tiene un contador de referencias
- Si realizamos aliasing el contador se incrementa y si liberamos se decrementa
- Cuando el contador vale 0 la zona de memoria se asigna a la zona libre
- Existen problemas cuando hay referencias circulares sin nombres. Al eliminar las con nombres el contador no se hace 0 y no se liberan.

Continuo on-the-fly

- Continuo on-the-fly es mediante un proceso paralelo al programa que se ejecuta y puede realizar tanto los algoritmos de copia como de mark-scan
- En la tabla resumen +/- significa bueno si hay mucha basura y -/+ bueno si hay poca. o significa neutro

Tabla resumen

	COPY	MARK–SCAN	CONTADOR–REF
EFICIENCIA TIEMPO	+/–	–/+	O
EFICIENCIA ESPACIO	–	+	–
LISTA ESPACIO LIBRE	NO	SI	SI
SIN OVERHEAD CON ESPACIO LIBRE	SI	SI	NO