Концепции языков программирования

Типы данных

Введение

- Тип данных — набор объектов и множество предопределенных операций над этими объектами
- Дескриптор — набор атрибутов переменной
- Объект представляет собой экземпляр пользовательского (абстрактного) типа данных
- Вопрос, возникающий при проектировании любого типа данных: Какие операции определены и как они специфицированы?

Элементарные типы данных

- Почти все языки программирования располагают набором примитивных типов данных
- Элементарные типы данных — типы, которые не определены в терминах других типов данных
- Некоторые элементарные типы данных просто отражают особенности аппаратного обеспечения
- Другие требуют некоторой программной поддержки

Элементарные типы данных

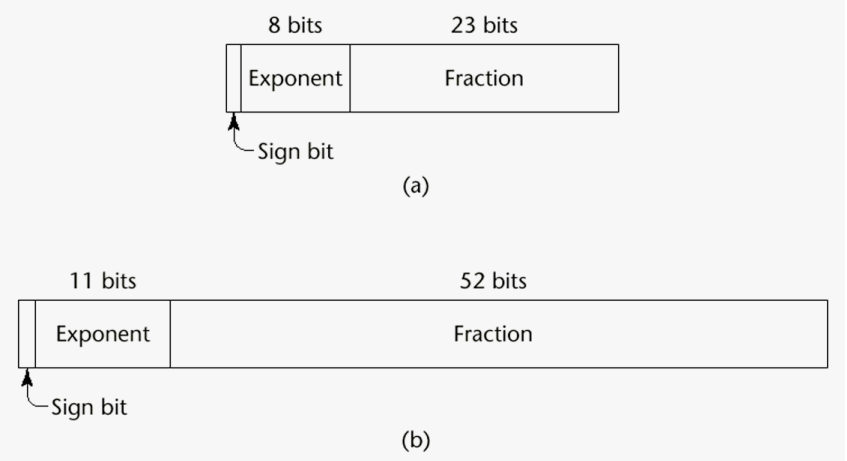
Целочисленные

- Почти всегда непосредственно отражают особенности аппаратного обеспечения
- В одном языке бывает до восьми различных целочисленных типов
- Знаковые целочисленные типы языка Java: `byte`, `short`, `int`, `long`

Элементарные типы данных

Числа с плавающей точкой

- Моделируют действительные числа, но только приблизительно
- Языки, предназначенные для научных вычислений, поддерживают, по крайней мере, два типа с плавающей точкой (например, `float` и `double`), иногда — больше
- Обычно, но не всегда, в точности соответствует аппаратному обеспечению
- IEE Floating-Point Standard 754



Элементарные типы данных

Десятичные числа

- Для бизнес-приложений (деньги)
  - Неотъемлемая часть языка COBOL
  - В C# есть тип данных decimal
- Хранит фиксированное количество десятичных
- Преимущество: точность
- Недостатки: ограниченная область значений, затраты памяти

Элементарные типы данных

Булевы типы

- Самые простые
- Область значений: два элемента, один для «истина», один для «ложь»
- Могут быть реализованы в виде битов, но обычно реализованы в виде байтов
- Преимущество: читабельность

- Хранятся как числовые коды
- Наиболее часто используемая кодировка — ASCII
- Альтернативная 16-ти битовая кодировка — Unicode
  - Включает символы большинства естественных языков
  - Первый язык программирования с поддержкой Unicode — Java
  - C# и JavaScript также поддерживают Unicode

Символьные строки

- Значения — последовательности символов
- Вопросы разработки:
  - Является ли примитивным типом или просто особым видом массива?
  - Должна ли длина строк определяться статически или динамически?

Операции над символьными строками

- Стандартные операции
  - Присваивание и копирование
  - Сравнение ($=$, $>$ и т.д.)
  - Конкатенация
  - Выделение подстроки
  - Сопоставление с образцом

Тип символьных строк в некоторых языках

- C и C++
  - Не является элементарным
  - Используются массивы `char` и библиотека функций, реализующих операции
- SNOBOL4 (язык для работы со строками)
  - Является элементарным
  - Многочисленные операции, включая сложные варианты сопоставления с образцом
- Java
  - Является элементарным (класс `String`)

Варианты длины строк

- Статическая: COBOL, класс `String` в Java
- Ограниченно-динамическая длина: C и C++
  - В языках, основанных на C, используется специальный символ для указания на то, что символы строки закончились, вместо того, чтобы непосредственно помнить длину строки
- Динамическая (нет максимума): SNOBOL4, Perl, JavaScript
- Ada поддерживает все три варианта длины строки

Анализ типа символьных строк

- Полезен при написании кода
- В качестве элементарного типа со статической длиной не являются дорогостоящими
- Динамическая длина удобна, но не всегда оправдана с точки зрения затрат

Реализация символьных строк

- Статическая длина: дескриптор во время компиляции
- Ограниченно-динамическая длина: может потребоваться дескриптор во время исполнения (но не в языках C и C++)
- Динамическая длина: нужен дескриптор во время исполнения; размещение в памяти и удаление из памяти — главная проблема реализации

Дескрипторы во время компиляции и исполнения

Статическая строка
Длина
Адрес

Дескриптор во время компиляции для статических строк

Ограниченно-динамическая строка
Максимальная длина
Текущая длина
Адрес

Дескриптор во время исполнения для ограниченно-динамических строк

Порядковые типы, определяемые пользователем

- Порядковый тип — такой, область возможных значений которого может быть легко сопоставлена множеству положительных целых чисел
- Примеры элементарных порядковых типов в языке Java
  - integer
  - char
  - boolean

Перечислимые типы

- Все возможные значения, являющиеся именованными константами, указаны в определении
- Пример из C#
  `enum days {mon, tue, wed, thu, fri, sat, sun};`
- Вопросы разработки
  - Может ли константа перечислимого типа появляться более, чем в одном определении типа, и, если да, как проверять тип этой константы при ее использовании?
  - Приводятся ли перечислимые значения к целым?
  - Приводится ли какой-либо другой тип к перечислимому типу?

Перечислимые типы

C

- перечислимые типы ведут себя как целочисленные типы
- допустимы арифметические операторы: `myDay++;`
- проверка диапазонов не осуществляется

C++

- константа перечислимого типа может появляться только в одном определении типа
- перечислимые значения приводятся к `int`: `myDay++;`
- другие типы не приводятся к перечислимым типам: `myDay = 4;`
- допускается явное приведение типов: `myDay = (Days)4;`
  - При явном приведении типов осуществляется проверка диапазонов

C#: как в C++, но без приведения к `int`

Перечислимые типы

Java 5.0 (2004)

- подкласс предопределенного класса `Enum`
- Методы: `values`, `ordinal`, ...
- `Enum` никогда не приводится к другим типам (таким как `int`)
- другие типы не приводятся к `Enum`
- арифметические операторы недопустимы
- осуществляются проверки диапазонов

Анализ перечислимых типов

- Способствуют читабельности: например, не нужно кодировать цвет числом
- Повышают надежность: например, компилятор может проверить
  - операции (не допустить сложение цветов)
  - Никакая переменная перечислимого типа не может получить значение за пределами области значений данного типа
  - Поддержка перечислимых типов в языках Ada, C# и Java 5.0 лучше, чем в C++, поскольку переменные перечислимого типа в этих языках не приводятся к целочисленным типам

Ограниченные типы

- Упорядоченная непрерывная подпоследовательность порядкового типа

Пример

12..18 — ограниченный подтип целочисленного типа

- В языке Ada:

```
type Days is (mon, tue, wed, thu, fri, sat, sun);
subtype Weekdays is Days range mon..fri;
subtype Index is Integer range 1..100;


Day1 Days;
Day2 Weekdays;
Day2 := Day1;
```

Анализ ограниченных типов

- Способствуют читабельности
  - Читателям ясно, что переменные ограниченного типа могут иметь значения только из определенной области
- Надежность
  - Присвоение переменной ограниченного типа значения, находящегося за пределами указанной области, обнаруживается как ошибка

Реализация пользовательских порядковых типов

- Перечислимые типы реализуют через целые числа
- Ограниченные типы реализуют как родительские типы с добавлением кода (осуществляемым компилятором), проверяющим допустимость присвоения значений переменным ограниченного типа

Массивы

- Массив — совокупность однородных элементов, в которой отдельный элемент идентифицируется по его позиции в совокупности относительно первого элемента.

Вопросы проектирования массивов

- Какие типы допустимы для индексов?
- Нужно ли осуществлять контроль диапазонов для ссылок с использованием индексов?
- Когда осуществлять связывание диапазонов индексов?
- Когда происходит выделение памяти?
- Каково максимальное число индексов?
- Можно ли инициализировать объекты-массивы?
- Допускаются ли какие-либо виды «срезов»?

Индексация массивов

- Индексация — это отображение индексов на элементы `array_name(index_value_list)` → `an element`
- Синтаксис индексов
  - FORTRAN, PL/I, ADA используют круглые скобки
    - В языке Ada круглые скобки используются специально, чтобы показать сходство между ссылками на элементы массивов и вызовами функций, так как те и другие являются отображениями
  - В большинстве других языков используются квадратные скобки

Типы индексов массивов

- Fortran, C, Java: только целочисленные
- Pascal, Ada: порядковый тип (целочисленный, булев, символьный, перечислимый)
- В C, C++, Perl и Fortran контроль диапазонов не осуществляется
- В Java, ML, C# контроль диапазонов осуществляется

Связывание индексов и категории массивов

- Статические диапазоны индексов: статическое связывание и статическое выделение памяти (до исполнения программы)
  - Преимущество: эффективность (нет динамического выделения памяти)
- Фиксированные стек-динамические: статическое связывание диапазонов индексов, но выделение памяти при обработке объявления переменной
  - Преимущество: эффективное использование памяти

Связывание индексов и категории массивов

Продолжение

- Стек-динамические: динамическое связывание диапазонов индексов и динамическое выделение памяти (во время исполнения)
  - Преимущество: гибкость (нет нужды знать размер массива до того, как массив будет использоваться)
- Фиксированные динамические: похожи на стек динамические — динамическое связывание и фиксация после выделения (но связывание осуществляется по явному запросу и память выделяется из кучи, а не из стека)

Связывание индексов и категории массивов

Продолжение

- Динамические: динамическое связывание диапазонов индексов и выделение памяти; диапазоны и память могут многократно изменяться во время исполнения
  - Преимущество: гибкость (массивы могут расти и сжиматься во время исполнения программы)

Связывание индексов и категории массивов

Продолжение

- Массивы C и C++, объявляемые с использованием модификатора `static`, являются статическими
- Массивы C и C++, объявляемые без использования модификатора `static`, являются фиксированными стек-динамическими
- Массивы Ada могут быть стек-динамическими
- В C и C++ есть фиксированные динамические массивы
- C# включает второй класс массивов — `ArrayLisrt`, который позволяет создавать динамические массивы
- Perl и JavaScript поддерживают динамические массивы

Связывание индексов и категории массивов

	Статические	Фиксированные стек-динамические	Стек-динамические	Фиксированные динамические	Динамические
Диапазон индексов	Статический	Статический	Динамический	Динамический	Динамический
Выделение памяти	Статическое	Динамическое (при обработке объявления)	Динамическое (при обработке объявления)	Динамическое (по запросу пользователя)	Динамическое (по запросу пользователя)
Изменение диапазона и памяти	Нет	Нет	Нет	Нет	Да
Пример	Статические массивы в C, C++	Нестатические массивы в C, C++	Массивы в языке Ada	Указатели в C, C++	ArrayList в C#; Perl, JavaScript

Инициализация массивов

- В некоторых языках есть возможность задать начальные значения элементам массива в момент выделения памяти

Пример (C, C++; аналогично в Java, C#)

```
int list[] = {4, 5, 7, 83};
```

Пример (Символьные строки в C и C++)

```
char name[] = "freddie";
```

Пример (Массивы строк в C и C++)

```
char * names[] = {"Bob" ,"Jake" ,"Joe"};
```

Пример (Инициализация объектов класса String в Java)

```
String[] names = {"Bob" ,"Jake" ,"Joe"};
```

Операции над массивами

- APL обладает наиболее мощным набором операций над массивами для работы с векторами и матрицами, включающим и унарные операторы (например, оператор, переворачивающий вектор)
- В языке Ada есть оператор присваивания, а также конкатенации
- Fortran располагает поэлементными операциями, т.е. такими, которые применяются к парам элементов массивов

Пример

Применение оператора + к двум массивам позволяет получить массив сумм пар соответствующих элементов этих массивов

Прямоугольные и рваные массивы

- Прямоугольный массив — многомерный массив, в котором все строки содержат одинаковое число элементов и все столбцы содержат одинаковое число элементов
  - `myArray[3, 7]`, C#
- Рваная матрица состоит из строк разной длины
  - Возможно, если многомерные массивы реализованы в виде массивов массивов
  - `myArray[3][7]`
  - C, C++, Java, C#

Сечения

- Сечение — подструктура массива; всего лишь механизм оформления ссылок на подструктуры
- Сечения полезны только в языках, располагающих операциями над массивами

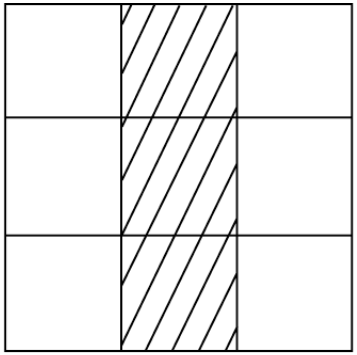
Примеры сечений

Пример (Fortran 95)

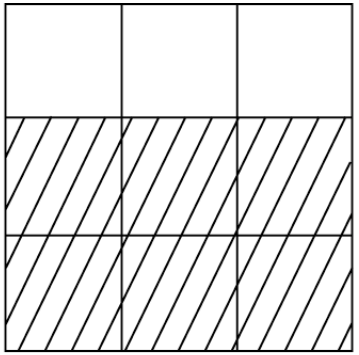
```
Integer,Dimension(10) :: Vector
Integer,Dimension(3,3) :: Mat
Integer,Dimension(3,3,4) :: Cube
```

`Vector(3 : 6)` — массив из четырех элементов

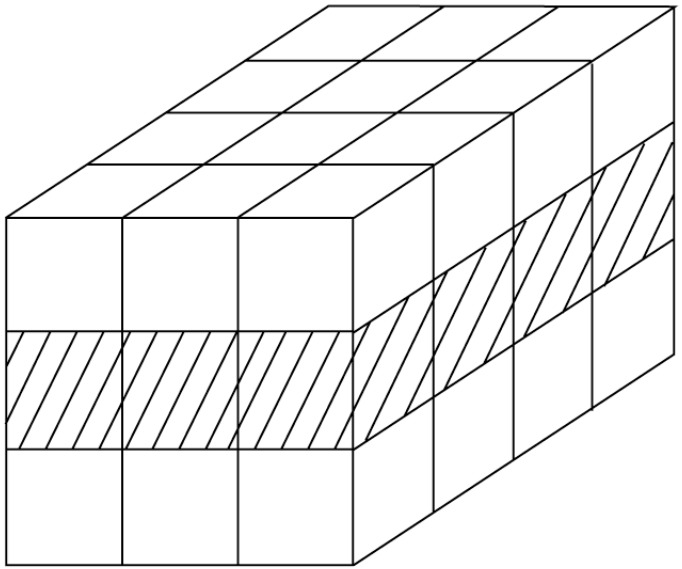
Примеры сечений в языке Fortran 95



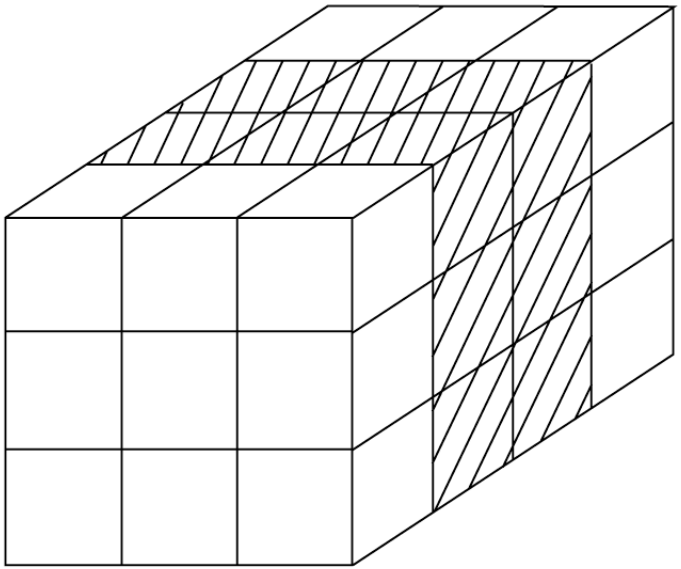
`MAT (1:3, 2)`



`MAT (2:3, 1:3)`



`CUBE (2, 1:3, 1:4)`



`CUBE (1:3, 1:3, 2:3)`

Реализация массивов

- Функция доступа отображает выражение с индексами на адрес в массиве
- Функция доступа для одномерных массивов:

  location(list[k]) = address(list[lower_bound]) +
  + ((k - lower_bound) * element_size)

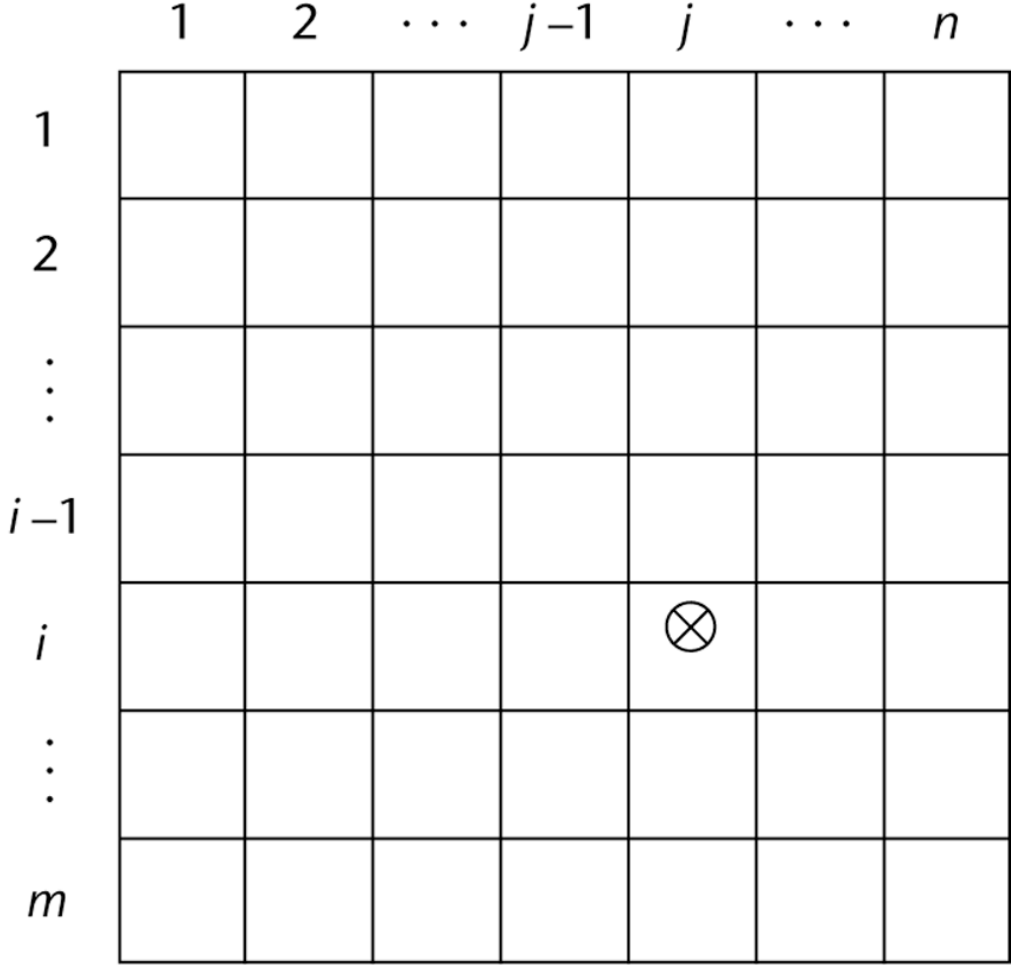
Доступ к элементам многомерных массивов

3	4	7
6	2	5
1	3	8

- Два общепринятых способа хранения массивов:
  - по строкам — используется в большинстве языков
  - 3, 4, 7, 6, 2, 5, 1, 3, 8
  - по столбцам — используется в языке Fortran
  - 3, 6, 1, 4, 2, 3, 7, 5, 8

Обнаружение элемента в многомерном массиве

Общий вид location(a[i, j]) = address(a[row_lb, col_lb]) +
(((i - row_lb) * n) + (j - col_lb)) * element_size



Дескрипторы при компиляции

Массив
Тип элемента
Тип индекса
Нижняя граница индексов
Верхняя граница индексов
Адрес

Одномерный массив

Многомерный массив
Тип элемента
Тип индекса
Число размерностей
Диапазон индексов 1
⋮
Диапазон индексов n
Адрес

Многомерный массив

Ассоциативные массивы

- Ассоциативный массив — неупорядоченная коллекция элементов-значений, проиндексированных соответствующим числом других элементов, называемых ключами
  - Ключи определяются пользователем и должны храниться
  - Каждый элемент: значение + ключ
- Вопросы проектирования:
  - В какой форме осуществляется ссылка на элементы?

Ассоциативные массивы в языке Perl

hashes

- Имена начинаются с %; литералы заключаются в скобки
  `%hi_temps = ("Mon" => 77,"Tue" => 79,"Wed" => 65,...);`
- При обращении к элементу индекс-ключ указывается в фигурных скобках
  `$hi_temps{"Wed"} = 83;`
- Элементы можно удалить из массива при помощи оператора `delete`
  `delete $hi_temps{"Tue"};`
- Удалить все элементы из массива
  `%hi_temps = ();`

Массивы и ассоциативные массивы

Массив : когда требуется обработать каждый элемент

Ассоциативный массив : когда нужно найти элемент в массиве

Типы-записи

- Запись — возможно разнородная совокупность элементов, в которой отдельные элементы различаются по именам
- Вопросы разработки:
  1. Каков синтаксис ссылок на поля записи?
  2. Допускаются ли эллиптические ссылки?

Определение записей

- В языке COBOL вложенные записи вводятся нумерацией уровней; в других языках используются рекурсивные определения
- Ссылки на поля записей
  1. COBOL
     <имя поля> OF <имя записи 1> OF ... OF <имя записи n>
  2. Другие языки (нотация с точкой)
     <имя записи 1>.<имя записи 2>. ... .<имя записи n>.<имя поля>

Определение записей в языке COBOL

- В языке COBOL вложенные записи вводятся нумерацией уровней; в других языках используются рекурсивные определения

Пример

```
01 EMP-REC.
   02 EMP-NAME.
      05 FIRST PIC X(20).
      05 MID PIC X(10).
      05 LAST PIC X(20).
   02 HOURLY-RATE PIC 99V99.
```

Определение записей в языке Ada

- Структуры записей определяются ортогональным образом

Пример

```
type Emp_Rec_Type is record
    First: String (1..20);
    Mid: String (1..10);
    Last: String (1..20);
    Hourly_Rate: Float;
end record;
Emp_Rec: Emp_Rec_Type;
```

Ссылки на записи

- Большинство языков используют нотацию с точкой
- Полные ссылки должны включать все имена записей
- В эллиптических ссылках имена записей могут быть опущены, если это не приводит к неоднозначности ссылки

Пример (COBOL)

Эллиптические ссылки на имя служащего:

- FIRST
- FIRST OF EMP-NAME
- FIRST OF EMP-REC

Операции над записями

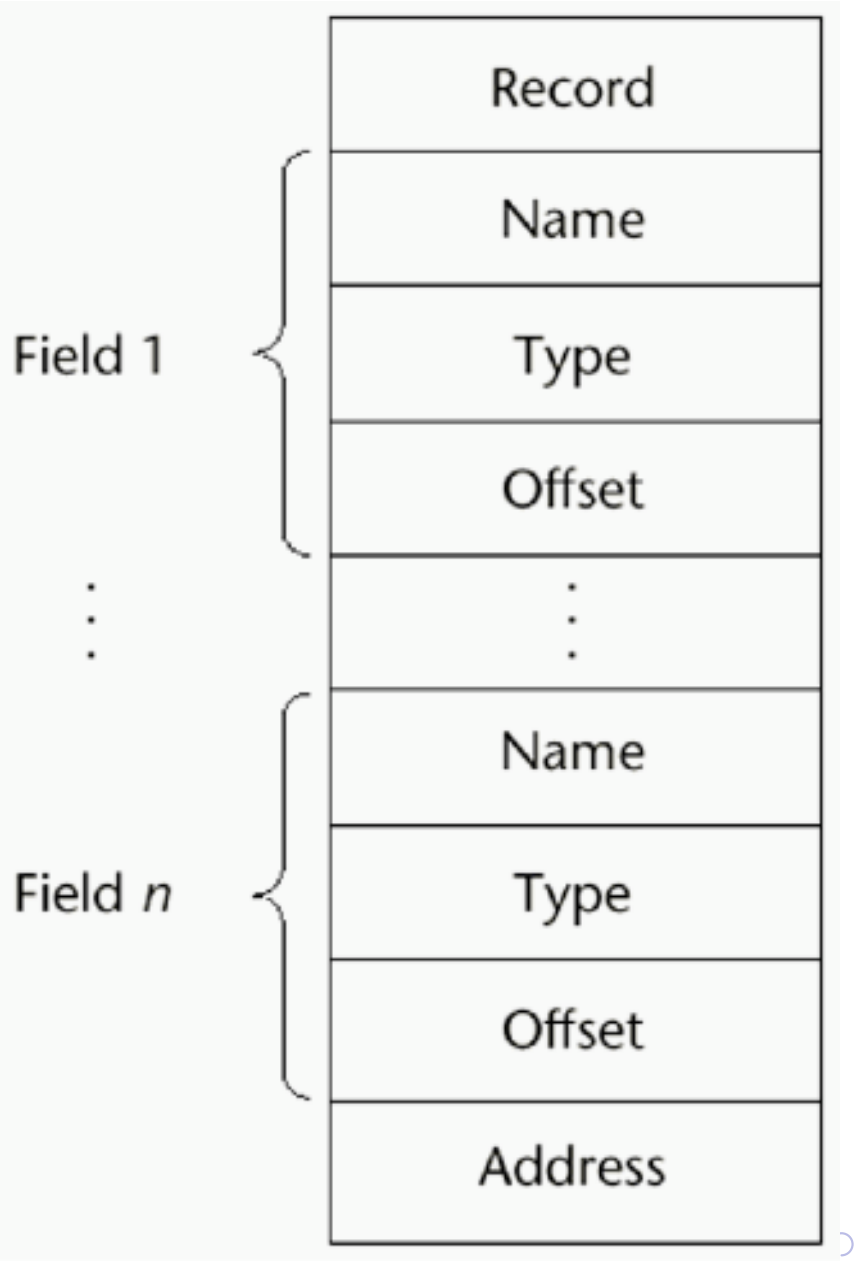
- Широко используется присваивание для операндов одного типа
- В языке Ada поддерживается сравнение записей
- Записи языка Ada могут быть инициализированы при помощи составных литералов
- MOVE_CORRESPONDING в языке COBOL
  - Копирует поле одной записи в соответствующее поле другой записи

Оценка и сравнение с массивами

- Прозрачная и безопасная структура
- Записи используются для разнородных наборов данных
- Доступ к элементам массивов гораздо медленнее доступа к полям записей, так как индексы вычисляются динамически (поля записей — статичны)
- Можно было бы использовать динамические индексы для доступа к полям записей, но это исключило бы проверку типов и было бы гораздо медленнее

Реализация типов-записей

С каждым полем связано смещение адреса относительно начала записи



Типы-объединения

- Объединение — это тип, переменные которого могут хранить значения разных типов в разные моменты выполнения программы
- Вопросы проектирования
  - Должна ли проверка типов быть обязательной?
  - Должны ли объединения использоваться только в составе записей?

Размеченные и свободные объединения

- В языках Fortran, C и C++ отсутствует проверка типов для объединений — свободные объединения
- Проверка типов для объединений требует, чтобы каждое объединение включало метку типа
  - Поддерживается в языке Ada

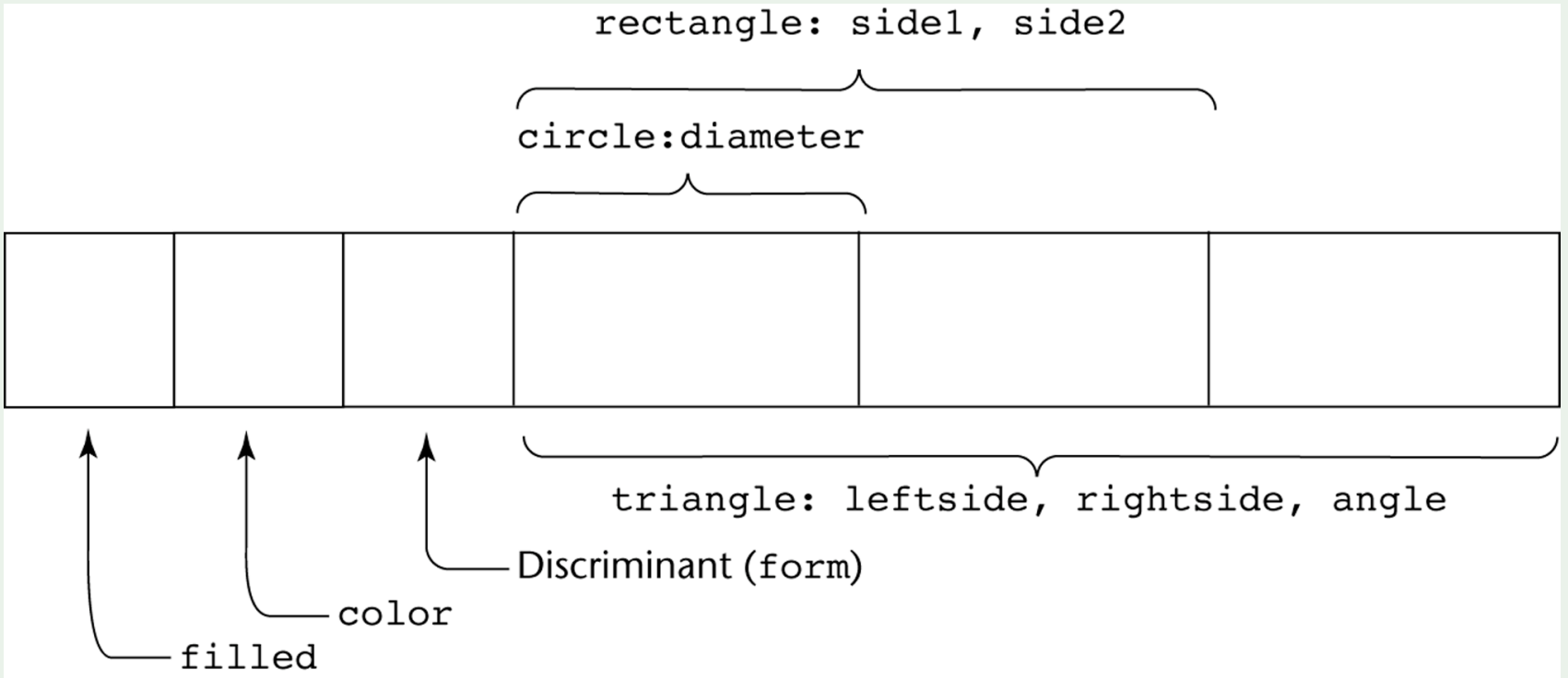
Типы-объединения языка Ada

Пример

```
type Shape is (Circle, Triangle, Rectangle);
type Colors is (Red, Green, Blue);
type Figure (Form: Shape) is record
    Filled: Boolean;
    Color: Colors;
    case Form is
        when Circle => Diameter: Float;
        when Triangle =>
            Leftside, Rightside: Integer;
            Angle: Float;
        when Rectangle => Side1, Side2: Integer;
    end case;
end record;
```

Типы-объединения языка Ada

Пример (Размеченное объединение из трех переменных)



Оценка объединений

- Потенциально небезопасная конструкция
  - без проверки типов
- Java и C# не поддерживают объединения
  - результат внимания, уделяемого вопросам безопасности в современных языках программирования

Указатели и ссылочные типы

- Область значений переменной типа указатель состоит из адресов в памяти и специального значения — nil
- Дают возможность косвенной адресации
- Дают возможность динамического управления памятью
- Указатель можно использовать для доступа к месту, где осуществляется динамическое выделение памяти (обычно это место называют кучей)

Вопросы проектирования указателей

- Каковы область видимости и время жизни переменной-указателя?
- Каково время жизни динамической переменной?
- Имеются ли ограничения относительно типов, на которые могут указывать указатели?
- Используются ли указатели для динамического управления памятью, косвенной адресации или и того, и другого?
- Должен ли язык поддерживать типы-указатели, ссылочные типы или и то, и другое?

Операции над указателями

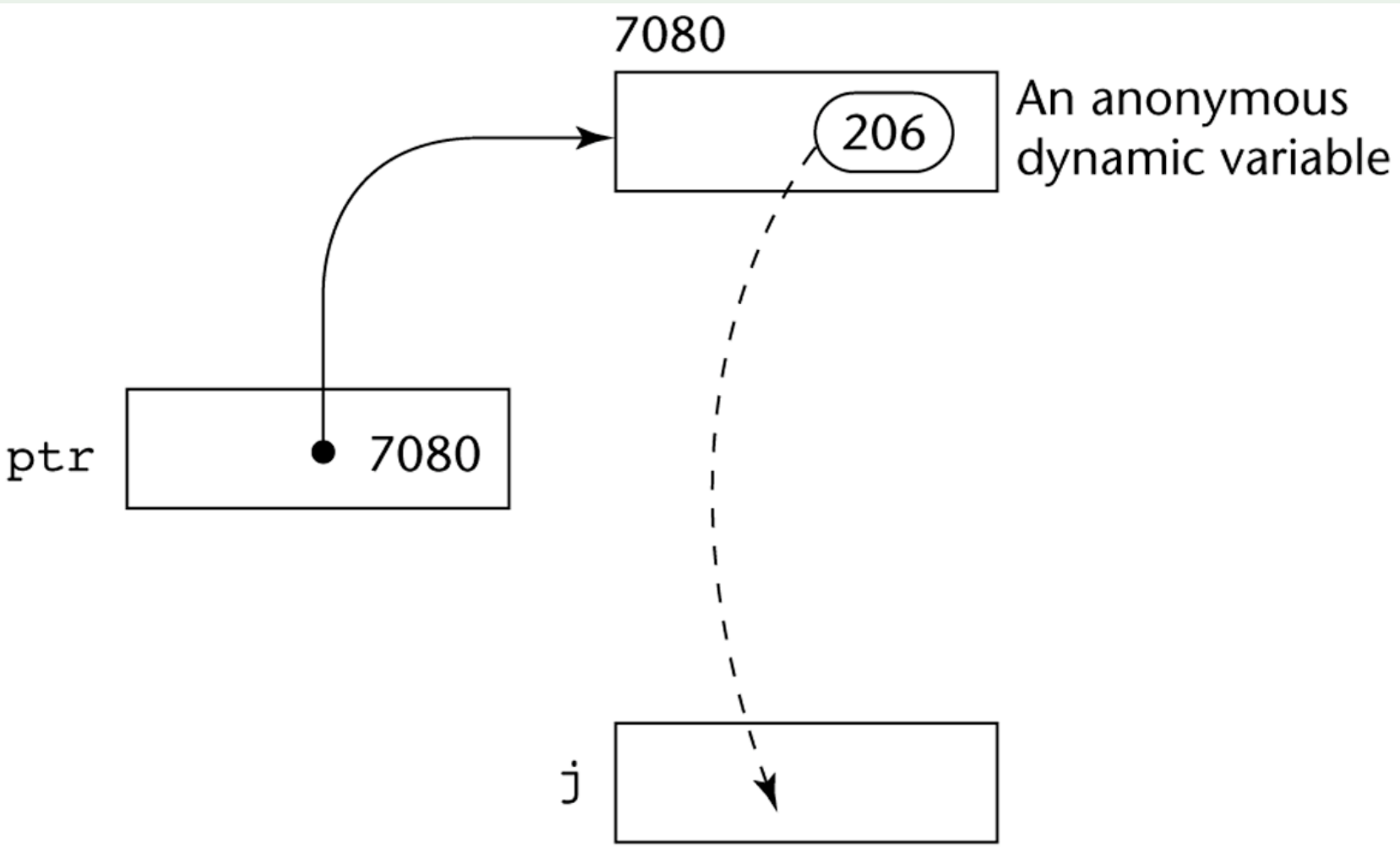
- Две основные операции: присваивание и разыменование
- Присваивание устанавливает значение переменной-указателя равным некоторому осмысленному адресу
- Разыменование возвращает значение, находящееся по адресу, являющемуся значением указателя
  - Разыменование может быть явным или неявным
  - В C++ используется явная операция разыменования, задаваемая при помощи *

Пример (Присвоить `j` значение, хранящееся по адресу, на которое указывает `ptr`)

```
j = *ptr
```

Разыменование указателя

Пример (`j = *ptr`)



Проблемы, связанные с указателями

- «Висячие» указатели (опасно)
  - Указатель указывает на динамическую переменную, которая была удалена из памяти
- «Потерянные» динамические переменные
  - Размещенная в памяти динамическая переменная, более недоступная программе (мусор)

Пример

- Указатель `p1` указывает на только что созданную динамическую переменную
- Затем указатель `p1` «переводят» на другую только что созданную динамическую переменную

Указатели в языке Ada

- Снижена вероятность появления висячих указателей, так как динамические объекты могут быть автоматически удалены из памяти в конце области видимости типа указателя
- Проблема потерянных динамических переменных присутствует в языке Ada

Указатели в языках C и C++

- Чрезвычайно гибкие, но требуют осторожности при использовании
- Указатели могут указывать на любую переменную независимо от того, когда она была размещена в памяти
- Используются для динамического управления памятью и адресации
- Поддерживается арифметика указателей
- Операторы явного разыменования и взятия адреса
- Не требуется, чтобы тип был фиксирован (`void*`)
- `void*` может указывать на любой тип, и на него распространяется проверка типов (не может быть разыменован)

Арифметика указателей в C и C++

```
float stuff[100];
float *p;
p = stuff;
```

*(p + 5) эквивалентно stuff[5] и p[5]
*(p + i) эквивалентно stuff[i] и p[i]

Указатели в языке Fortran 95

- Указатели могут указывать как на переменные, размещенные в куче, так и на переменные, размещенные не в куче
- Неявное разыменование
- Указатели могут указывать только на переменные с атрибутом `TARGET`
- Атрибут `TARGET` указывается в объявлении переменной: `INTEGER, TARGET :: NODE`

Ссылки

- В C++ есть указатели специального вида, называемые ссылками и используемые, главным образом, в качестве формальных параметров
  - Преимущества передачи параметров по ссылке и по значению
- Java расширяет ссылочные переменные C++, которые полностью заменяют указатели
  - Ссылки ссылаются на экземпляры классов
- В C# есть как ссылки, аналогичные ссылкам из Java, так и указатели C++

Анализ указателей

- Проблемы: висячие указатели и потерянные объекты, а также управление памятью
- Указатели сравнивают с `goto` — они увеличивают количество ячеек, на которые может ссылаться переменная
- Указатели или ссылки необходимы для динамических структур данных — поэтому без них невозможно спроектировать язык

Реализация указателей

- В больших компьютерах используются единичные значения
- В интеловских микропроцессорах используются сегмент и отступ

Проблема висячих указателей

- Надгробие: дополнительная ячейка в куче, являющаяся указателем на динамическую переменную
  - Настоящая переменная-указатель указывает только на надгробия
  - Когда динамическая переменная удаляется из памяти, надгробие остается, но устанавливается равным nil
  - Дорого с точки зрения времени и пространства
- Система замков и ключей: значения указателей — пары (ключ, адрес)
  - Динамическая переменная дополнена ячейкой с целочисленным значением замка
  - Когда динамическая переменная размещается в памяти, создается значение замка и оно помещается в ячейку замка и в ячейку ключа указателя

Управление кучей

- Очень сложный процесс времени выполнения
- Ячейки одного размера и ячейки переменного размера
- Два подхода к утилизации мусора
  - Счетчики ссылок (энергичный подход): постепенная уборка
  - Сборка мусора (ленивый подход): утилизация происходит, когда список доступных ячеек для переменных становится пустым

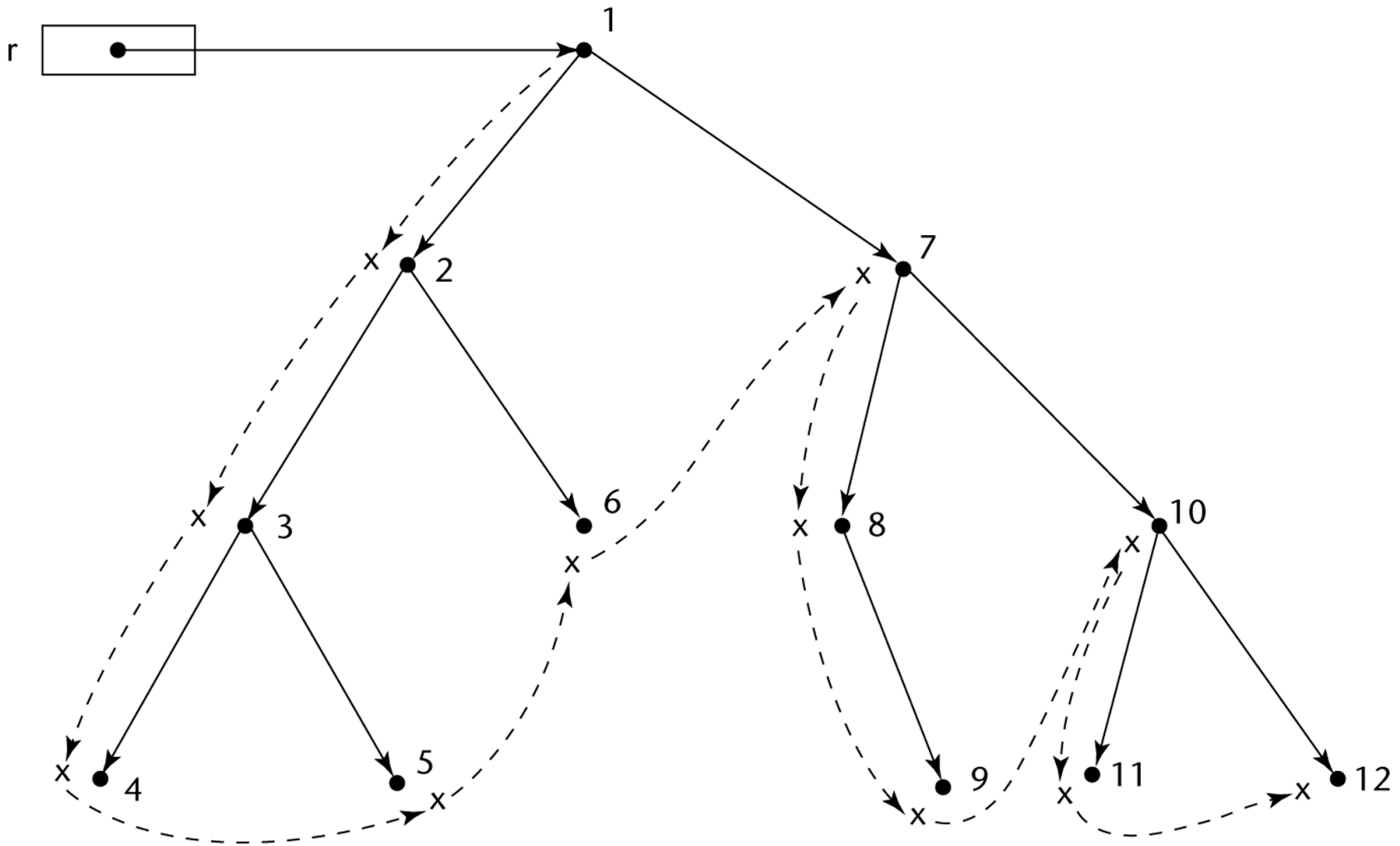
Счетчик ссылок

- Счетчики ссылок: счетчик для каждой ячейки, в котором хранится число указателей, на данный момент указывающих на эту ячейку
  - Недостатки: дополнительное пространство; увеличение времени выполнения; сложности, возникающие, если ячейки соединены «по кругу»

Сборка мусора

- Выделение ячеек памяти и отсоединение указателей происходит, когда требуется; когда начинается сборка мусора
  - В каждой ячейке кучи есть дополнительный бит, используемый алгоритмом сборки
  - Сначала каждая ячейка помечается как мусор
  - Отслеживаются все указатели, и соответствующие ячейки в куче помечаются как не являющиеся мусором
  - Все мусорные ячейки возвращаются в список доступных ячеек
  - Недостатки: когда она нужна больше всего, она работает хуже всего (занимает больше всего времени, когда программа нуждается в большей части ячеек кучи)

Алгоритм разметки



Dashed lines show the order of node_marking

Ячейки разных размеров

- Все трудности, связанные с ячейками одного размера, и дополнительные трудности
- Требуются в большинстве языков программирования
- При использовании сборки мусора возникают дополнительные проблемы
  - Затруднена начальная установка индикаторов всех ячеек в куче
  - Процесс разметки нетривиален
  - Поддержка списка доступных ячеек — еще один источник неэффективности

Резюме

- Типы данных, имеющиеся в языке, во многом определяют его стиль и полезность
- Элементарные типы данных в большинстве императивных языков включают числовые, символьные и булевы типы
- Пользовательские перечислимые и ограниченные типы удобны в использовании и повышают читабельность и надежность программ
- Массивы и записи включены в большинство языков
- Указатели обеспечивают гибкость адресации и позволяют осуществлять динамическое управление памятью