Концепции языков программирования

Распределение памяти

Распределение памяти

Распределение памяти

Статическое выделение пямяти

- Выделение памяти на этапе загрузки программы
- Данные, находящиеся в статически выделенной ячейке памяти, доступны при повторных вызовах программного модуля, в котором определена соответствующая переменная
- Обычно статически размещаются в памяти
  - объектные константы, значение которых известно во время компиляции
  - таблицы виртуальных методов классов

Статические переменные

- Время жизни статической переменной совпадает с временем работы программы
- В некоторых языках (Паскаль) все локальные переменные являются стек-динамическими, а все глобальные — статическими

Статические переменные

Язык C

Статические глобальные переменные объявляются на верхнем уровне файла; область видимости — файл

Статические локальные переменные объявляются внутри функции; область видимости — функция

Статические данные-члены — переменные класса

Распределение памяти

Автоматические переменные

Автоматическая переменная — переменная со статической областью видимости, которая автоматически

- размещается в памяти, когда поток выполнения входит в ее область видимости,
- удаляется из памяти, когда поток выполнения (окончательно) покидает ее область видимости.

- Размещаются в записи активации процедуры, в которой они объявлены
- Обеспечивают поддержку рекурсии
- Могут быть размещены в регистрах процессора

Автоматические переменные

в языках программирования

C++ Любая переменная, объявленая внутри некоторого блока кода, по умолчанию является автоматической (ключевое слово auto)

- Значение не определено, до явного присвоения
- Ключевое слово register — рекомендация компилятору размещать переменную в регистре процессора

Java Обязательная инициализация

Perl Оператор my

- Значение по умолчанию для скалярных переменных — undef
- Значение по умолчанию для массивов — ()
- Оператор local не создает новых переменных, но сообщает новое временное значение существующей переменной

Распределение памяти

Buddy memory allocation

Система двойников

Память последовательно делится на два блока одинакового размера с тем, чтобы выделить объем памяти, максимально приближенный к требуемому.

Buddy memory allocation

Система двойников

- Относительно простая реализация
- Внешняя фрагментация незначительна

  Внешняя фрагментация: памяти достаточно, но она разбита на слишком маленькие блоки
- Возможна внутренняя фрагментация

  Внутренняя фрагментация: выделяется больше памяти, чем необходимо

Buddy memory allocation

Система двойников

- Память выделяется блоками размером 2^x
  - Для x заданы верхняя и нижняя границы u и l

Buddy memory allocation

Система двойников

- Память выделяется блоками размером 2^x
  - Для x заданы верхняя и нижняя границы u и l
- Выделение памяти:
  - Найти блок минимального подходящего размера 2^x
  - Если блок найден, выделить память
  - Делить блок большего размера пополам, пока не будет получен блок желаемого размера

Buddy memory allocation

Система двойников

- Память выделяется блоками размером 2^x
  - Для x заданы верхняя и нижняя границы u и l
- Выделение памяти:
  - Найти блок минимального подходящего размера 2^x
  - Если блок найден, выделить память
  - Делить блок большего размера пополам, пока не будет получен блок желаемого размера
- Освобождение памяти:
  - Освободить блок памяти
  - Если соседний блок (двойник) свободен, объединить два блока в один
  - Повторять предыдущий шаг, пока это возможно

Buddy memory allocation

Система двойников

	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K
$t = 0$	1024K															
$t = 1$	A-64K	64K	128K		256K				512K							
$t = 2$	A-64K	64K	B-128K		256K				512K							
$t = 3$	A-64K	C-64K	B-128K		256K				512K							
$t = 4$	A-64K	C-64K	B-128K		D-128K		128K		512K							
$t = 5$	A-64K	64K	B-128K		D-128K		128K		512K							
$t = 6$	128K		B-128K		D-128K		128K		512K							
$t = 7$	256K				D-128K		128K		512K							
$t = 8$	1024K															

Buddy memory allocation

Система двойников

	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K
$t = 0$	1024K															
$t = 1$	A-64K	64K	128K		256K				512K							
$t = 2$	A-64K	64K	B-128K		256K				512K							
$t = 3$	A-64K	C-64K	B-128K		256K				512K							
$t = 4$	A-64K	C-64K	B-128K		D-128K		128K		512K							
$t = 5$	A-64K	64K	B-128K		D-128K		128K		512K							
$t = 6$	128K		B-128K		D-128K		128K		512K							
$t = 7$	256K				D-128K		128K		512K							
$t = 8$	1024K															

- Эффективное освобождение памяти — $\log_2(u/l)$ операций объединения
- Обычно реализуется с использованием бинарного дерева
- Внутренняя фрагментация
  - Решение — slab allocation (многократное использование блока памяти для размещения объектов одного типа)

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

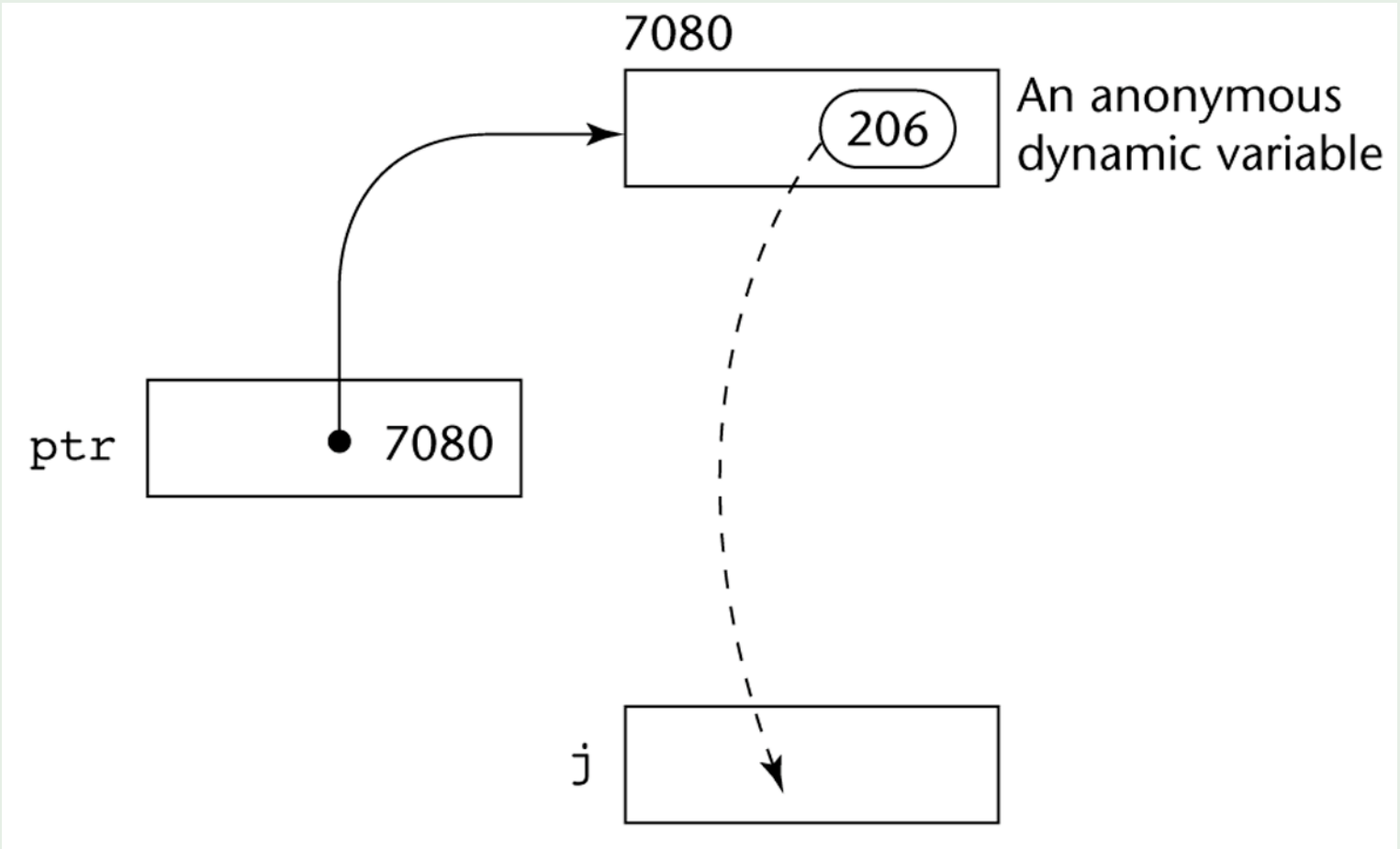
- Две основные операции: присваивание и разыменование
- Присваивание устанавливает значение переменной-указателя равным некоторому осмысленному адресу
- Разыменование возвращает значение, находящееся по адресу, являющемуся значением указателя
  - Разыменование может быть явным или неявным
  - В C++ используется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

Пример

```
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:

Пример

```
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 область применения таких переменных-ссылок, полностью заменяя ими 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 полноценный язык программирования

Реализация указателей

- В больших компьютерах адрес — одно число
- В интеловских микропроцессорах используются сегмент и отступ

Проблема висячих указателей

Надгробие: дополнительная ячейка в куче, являющаяся указателем на динамическую переменную

- Указатели указывают только на надгробия
- При удалении динамической переменной из памяти надгробие остается, но устанавливается равным nil
- Дорого с точки зрения времени и памяти

Проблема висячих указателей

Надгробие: дополнительная ячейка в куче, являющаяся указателем на динамическую переменную

- Указатели указывают только на надгробия
- При удалении динамической переменной из памяти надгробие остается, но устанавливается равным nil
- Дорого с точки зрения времени и памяти

Система замков и ключей: значения указателей — пары (ключ, адрес)

- Динамическая переменная дополнена ячейкой с целочисленным значением замка
- При размещении динамической переменной в памяти создается значение замка, которое помещается в ячейку замка и в ячейку ключа указателя

Управление динамической памятью

- Сложный процесс, происходящий во время выполнения программы
- Ячейки постоянного размера и ячейки переменного размера
- Два подхода к освобождению памяти

  Счетчики ссылок (энергичный подход): инкрементное освобождение памяти

  Сборка мусора (ленивый подход): освобождение памяти происходит, когда доступная память исчерпана

Счетчик 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

- Вид автоматического управления памятью — поиск и освобождение неиспользуемой памяти
- Используется вместе или вместе с ручным управлением памятью
- Используется только для освобождения памяти, но не других ресурсов

Сборка мусора

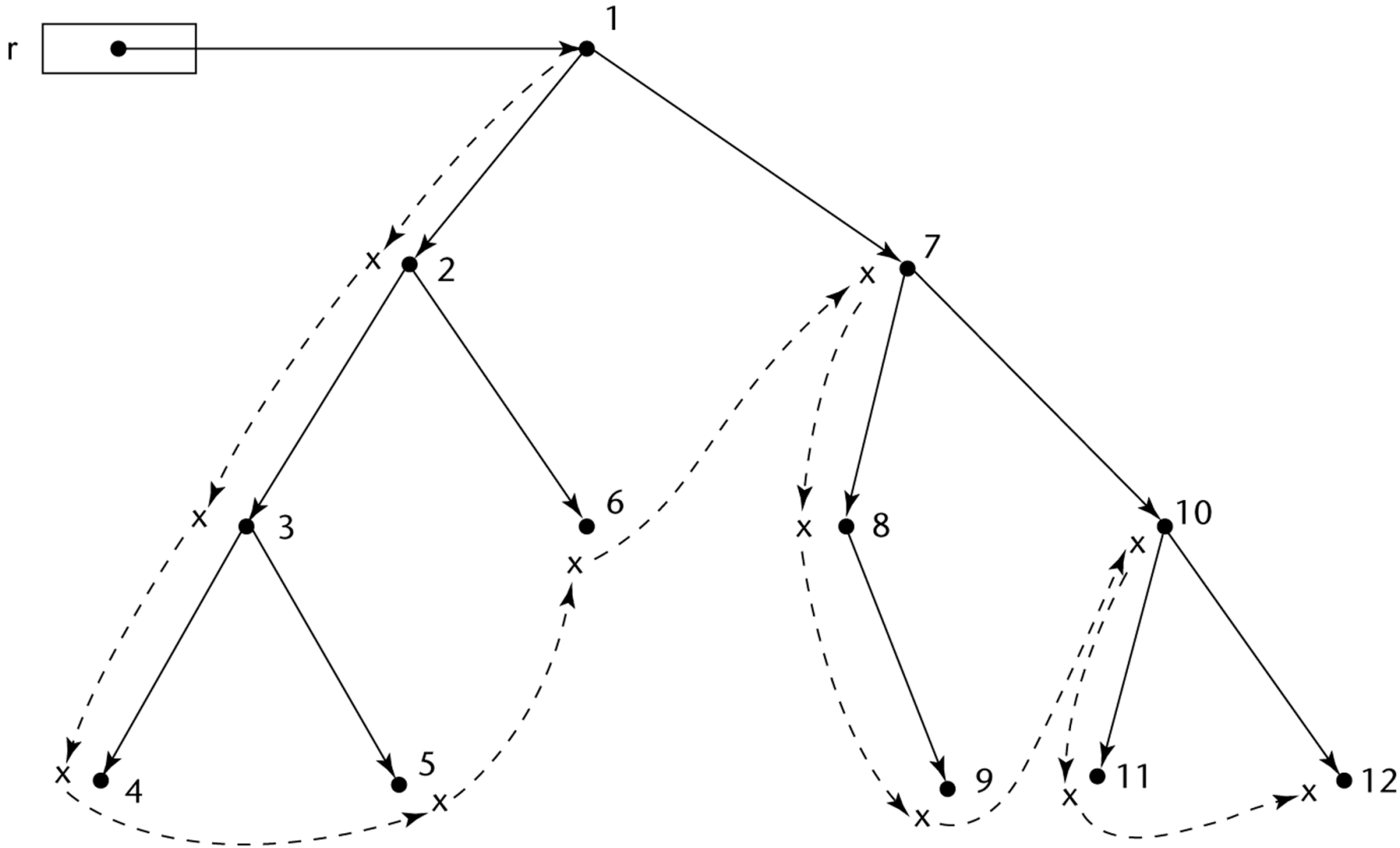
- Выявить недостижимые объекты
- Освободить память, занимаемую этими объектами

Сборка мусора

- Выделение ячеек памяти и открепление от них указателей происходит по мере необходимости
  - В каждой ячейке кучи есть дополнительный бит, используемый алгоритмом сборки
  - Сначала каждая ячейка помечается как мусор
  - Отслеживаются все указатели, и соответствующие ячейки памяти помечаются как используемые
  - Все мусорные ячейки возвращаются в список доступных ячеек

Недостатки: когда сборка мусора нужна больше всего, она работает хуже всего (занимает больше всего времени, когда программа нуждается в наибольшем количестве памяти)

Алгоритм разметки



Ячейки переменного размера

- Все трудности, связанные с ячейками постоянного размера, и дополнительные трудности
- Требуются в большинстве языков программирования
- Проблемы, возникающие при использовании сборки мусора
  - Затруднена начальная установка индикаторов всех ячеек памяти
  - Процесс разметки нетривиален
  - Поддержка списка доступных ячеек — еще один источник неэффективности