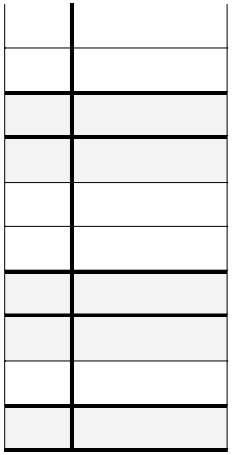
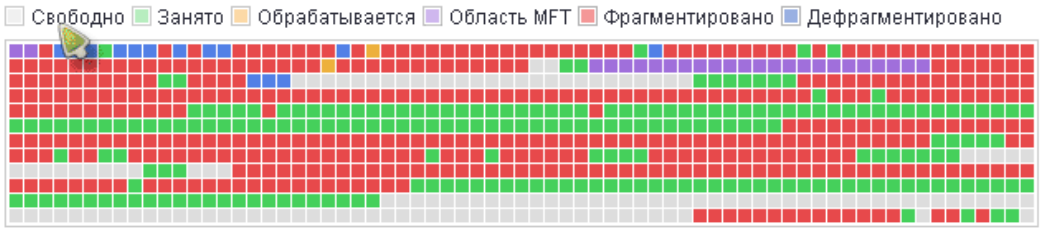
Карпов В.Э.

Сборка мусора

Определение

- **Сборка мусора** (*garbage collection*) — одна из форм автоматического управления памятью.
- Специальный код, называемый *сборщиком мусора* (*garbage collector*), периодически освобождает память, удаляя объекты, которые уже не будут востребованы приложением.





История

- Сборка мусора была впервые применена Джоном Маккарти в 1959 году в системе Лисп. Далее она применялась в других системах программирования и языках, преимущественно — в функциональных и логических (структура таких языков делает крайне неудобным отслеживание времени жизни объектов в памяти и ручное управление ею).
- В процедурных и объектных языках сборка мусора не использовалась до конца 1980-х гг.
- Со второй половины 1990-х гг. – почти во всех новых языках механизм сборки мусора вводится либо как единственный, либо как один из доступных механизмов управления динамической памятью (Oberon, Java, Python, Ruby, Perl, C#, D и др.)

Проблемы ручного управления памятью

Сущность ручного управления памятью:

- Для создания объекта в динамической памяти программист явно вызывает команду выделения памяти. Эта команда возвращает указатель на выделенную область памяти, который сохраняется и используется для доступа к ней.
- До тех пор, пока созданный объект нужен для работы программы, программа обращается к нему через ранее сохранённый указатель.
- Когда надобность в объекте проходит, программист явно вызывает команду освобождения памяти, передавая ей указатель на удаляемый объект.

```
Obj *p;
p = new Obj;
p->Eval();
delete p;
```

В любом языке, допускающем создание объектов в динамической памяти, потенциально возможны две проблемы: *висячие ссылки* и *утечки памяти*.

Висячие ссылки

- Висячая ссылка (*dangling pointer*) — это оставшаяся в использовании ссылка на объект, который уже удалён. После удаления объекта все сохранившиеся в программе ссылки на него становятся «висячими».
- Память, занимаемая ранее объектом, может быть передана ОС и стать **недоступной**, или быть использована для **размещения нового объекта** в той же программе. В первом случае попытка обратиться по «повисшей» ссылке приведёт к срабатыванию механизма защиты памяти и аварийной остановке программы, а во втором — к непредсказуемым последствиям.
- Появление висячих ссылок обычно становится следствием неправильной оценки времени жизни объекта.

```
Obj *p1, *p2;
p1 = new Obj;
p2 = p1;
p1->Eval();
delete p1;
...
p2->Eval();
```

Утечка памяти

Утечка памяти (*memory leak*)

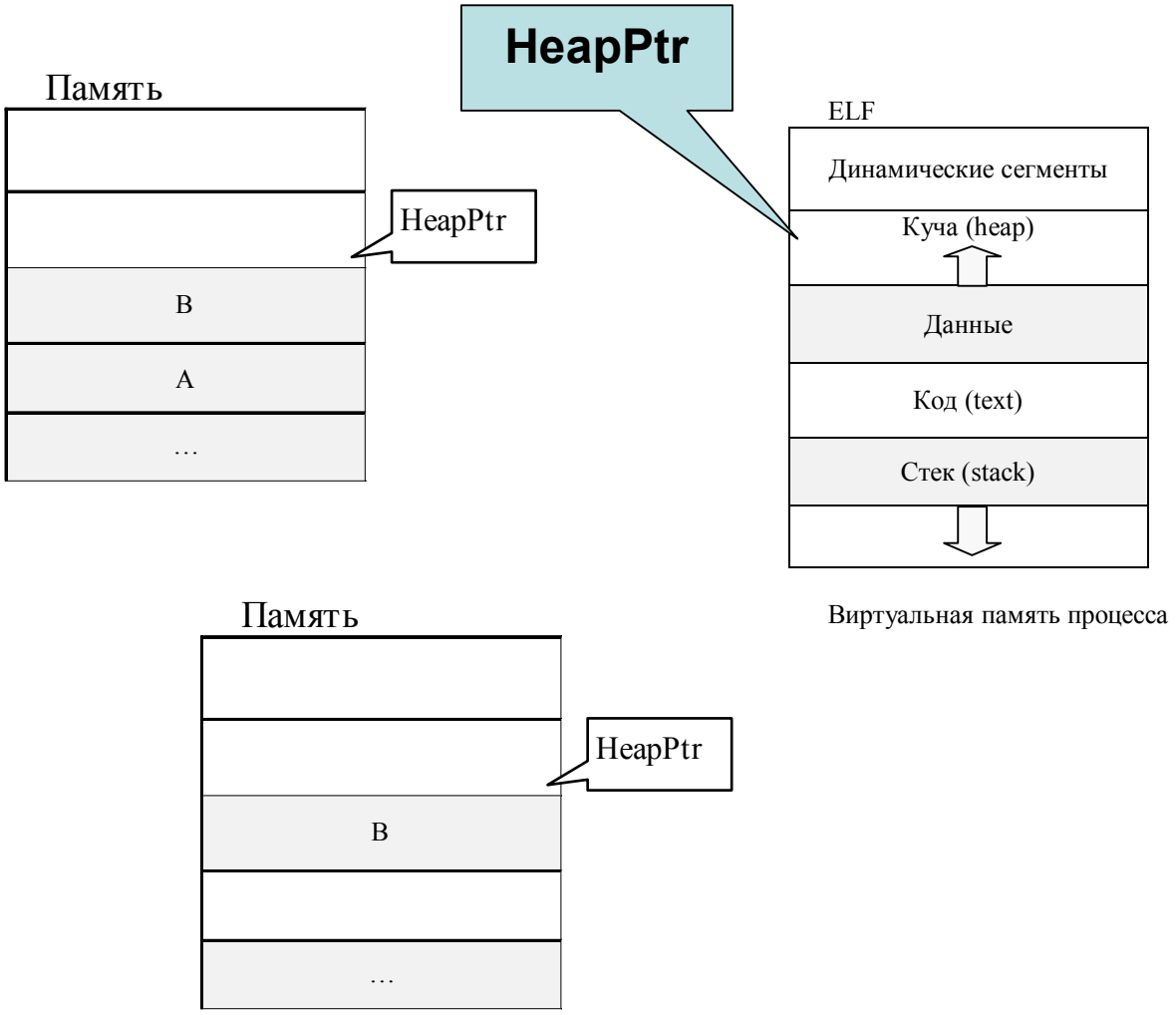
- Создав объект в динамической памяти, программист может не удалить его после завершения использования. Если ссылающейся на объект переменной будет присвоено новое значение и на объект нет других ссылок, он становится программно недоступным, но продолжает занимать память, поскольку команда его удаления не вызывалась.

```
Obj *p;
p = new Obj;
p->Eval();
...
p = new Obj;
delete p;
```

Более простая ситуация утечки

```
for(...)
{
  a = new A;
  b = new B;

  delete a;
  delete b;
}
...
c = new C; //?
```



Основные принципы сборки мусора

- Сборка мусора (СМ) — технология, позволяющая:
  - упростить программирование, избавив программиста от необходимости вручную удалять объекты, созданные в динамической памяти,
  - устранить ошибки, вызванные неправильным ручным управлением памятью.
- В системе со СМ обязанность освобождения памяти от объектов, которые больше не используются, возлагается на **среду исполнения программы**.
- Для осуществления СМ состав среды исполнения включается специальный программный модуль - *сборщик мусора*.
- Этот модуль периодически запускается, определяет, какие из созданных в динамической памяти объектов более не используются и освобождает занимаемую ими память.

Периодичность

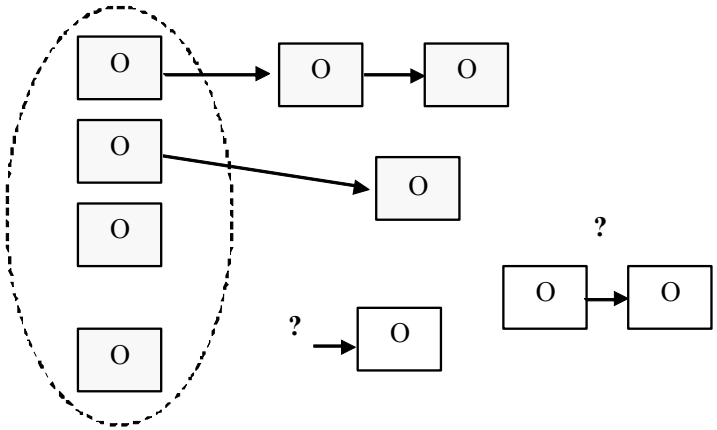
Периодичность запуска СМ определяется особенностями системы:

1. СМ работает в фоновом режиме, запускаясь при неактивности программы.
2. СМ запускается безусловно, прерывая выполнение программы, когда очередную операцию выделения памяти оказывается невозможно выполнить из-за того, что вся доступная память исчерпана.

Хотя, в общем случае, невозможно точно определить момент, когда объект был использован в последний раз и больше не нужен, СМ используют **консервативные** оценки, позволяющие определить, что в будущем объект **гарантированно** не будет использоваться.

Достижимость объекта

Критерием того, что объект ещё используется, является **наличие ссылок** на него. Если в системе нет больше ссылок на данный объект, то он больше не может быть использован программой и может быть удалён.



Рекурсивное определение достижимого объекта:

- определённое множество объектов считается достижимым изначально — *корневые объекты* (все глобальные переменные и объекты, на которые есть ссылки в стеке вызовов);
- любой объект, на который есть ссылка из достижимого объекта тоже считается достижимым.

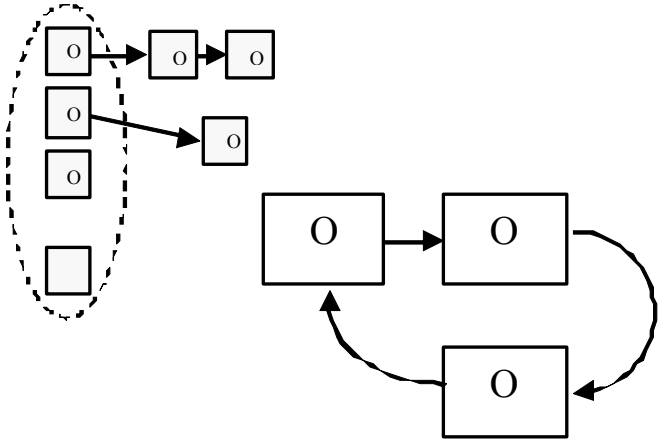
Такое определение не является теоретически наилучшим, т.к. в число достижимых, согласно ему, попадают и те объекты, которые уже никогда не будут использованы, но на которые пока ещё существуют ссылки.

Оптимальным было бы считать недостижимым объект, к которому в процессе дальнейшей работы программы не будет ни одного обращения, однако выявление таких объектов невозможно, поскольку сводится к алгоритмически неразрешимой задаче об остановке.

Алгоритм выставления флагов

Простой алгоритм определения достижимых объектов, «алгоритм пометок» (Mark and Sweep) :

- для каждого объекта хранится бит, указывающий, достижим ли этот объект из программы или нет;
- изначально все объекты, кроме корневых, помечаются как недостижимые;
- рекурсивно просматриваются и помечаются как достижимые объекты ещё не помеченные и до которых можно добраться из корневых объектов по ссылкам;
- те объекты, у которых бит достижимости не был установлен, считаются недостижимыми.



Примечание. Если два или более объектов ссылаются друг на друга, но ни на один из этих объектов нет других ссылок, т.е. имеется циклическая ссылка, то вся группа считается недостижимой. Эта особенность алгоритма позволяет гарантированно удалять группы объектов, использование которых прекратилось, но в которых имеются ссылки друг на друга (острова изоляции - «islands of isolation»).

Алгоритм подсчёта ссылок

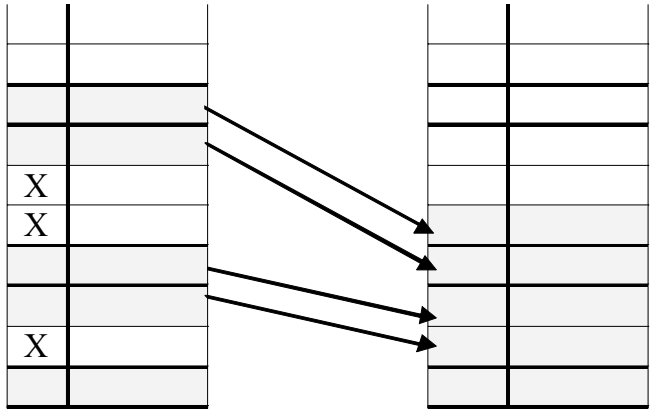
Обычный подсчёт ссылок на объекты.

- Его использование замедляет операции присваивания ссылок, но зато определение достижимых объектов тривиально — это все объекты, значение счётчика ссылок которых превышает нуль.
- Без дополнительных уточнений этот алгоритм, в отличие от предыдущего, **не удаляет** циклически замкнутые цепочки вышедших из употребления объектов, сохранивших ссылки друг на друга.

Стратегии сборки мусора

После определения недостижимых объектов:

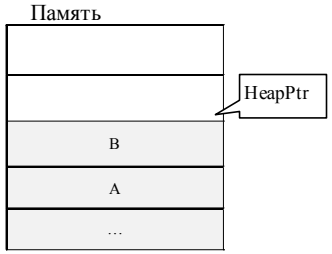
- Освободить занимаемую ими память и оставить остальное как есть (**неперемещающая стратегия**)
- После освобождения памяти переместить имеющиеся объекты обновив вместе с этим все ссылки на них (**перемещающая стратегия**)



Особенности стратегий

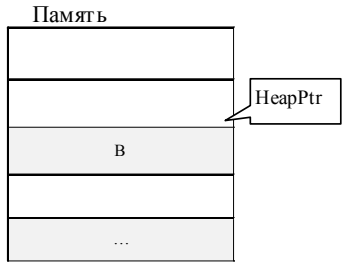
1. Скорость выделения и освобождения памяти

- **Неперемещающий** СМ быстро освобождает память, но тратит больше времени на её выделение (память фрагментируется и необходимо найти в карте памяти нужное количество блоков подходящего размера).
- **Перемещающий** СМ требует больше времени на этапе сборки мусора (тратится время на дефрагментацию памяти и изменение всех ссылок), зато просто и быстро выделяет память (имеется указатель на границу занятой и свободной областей памяти).



2. Скорость доступа к объектам в динамической памяти

- Объекты, поля которых используются совместно, **перемещающий** СМ может размещать в памяти недалеко друг от друга. Тогда они вероятнее окажутся в кэше одновременно => уменьшение количества обращений к медленной ОП.



3. Совместимость с инородным кодом

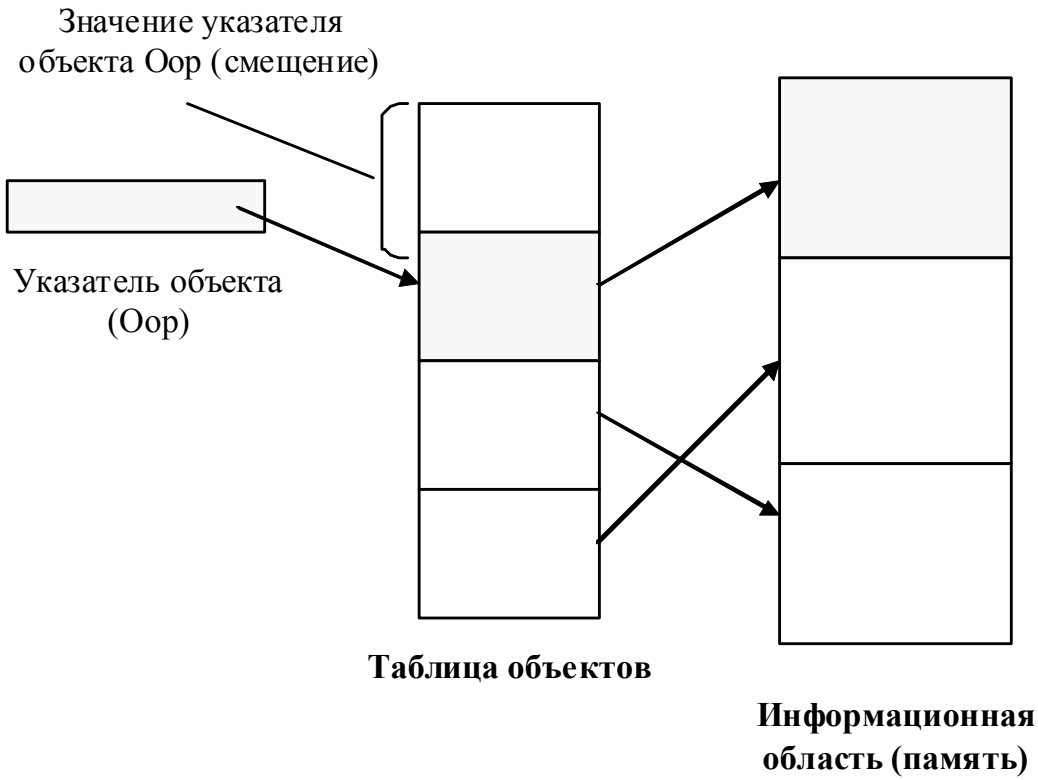
- **Перемещающий** СМ имеет проблемы при использовании кода, который не контролируется системой СМ. Указатель на память, выделенную в системе с **неперемещающим** сборщиком, можно просто передать инородному коду для использования.

Периодичность запуска сборщика мусора

- **Работа в фоновом режиме**, запускаясь при неактивности программы (например, когда программа простаивает, ожидая ввода данных пользователем).
- **Безусловный запуск**. Прерывание выполнения программы, когда очередную операцию выделения памяти невозможно выполнить из-за того, что вся доступная память исчерпана.

Сборка мусора в Smalltalk

- Реализована методом счетчика ссылок: если значение счетчика, связанного с некоторым объектом, становится равным нулю, объект переводится в список "мусора".
- Процедура сборки мусора выполняется, когда:
  - свободная память исчерпывается полностью
  - наблюдается простой системы.
- **Недостаток** метода - большие накладные расходы на ведение счетчиков.



Проблемы использования СМ

СМ не освобождает программиста от всех проблем управления памятью.

Освобождение других ресурсов, занятых объектом

- Помимо динамической памяти, объект может владеть и другими ресурсами. Файлы. Деструкторы и финализаторы.

Утечка памяти

- Ссылка на неиспользуемый объект может сохраниться в другом объекте, который используется и удерживает ненужный объект в памяти.

- Чтобы устранить подобные проблемы, среда исполнения может поддерживать специальное средство — так называемые *слабые ссылки*.

Достоинства и недостатки

- СМ безопаснее, поскольку она предотвращает утечки памяти и возникновение висячих ссылок из-за несвоевременного удаления объектов.
- Упрощение процесса программирования.
- Системы СМ менее эффективны как по скорости, так и по объёму используемой памяти.
- В системах со СМ сложнее реализуются низкоуровневые алгоритмы, требующие прямого доступа к оперативной памяти.
- Поддержка автоматического вызова деструкторов и использование «умных ссылок» (отслеживание количества ссылок на объект непосредственно в нём и автоматическое удаление при удалении последней ссылки) может быть более эффективным, чем СМ.
- СМ эффективна тогда, когда используются динамически изменяемые структуры данных (списки, деревья, графы). Например, Лисп или Пролог.
- Сборка мусора – затратный механизм (и по памяти, и по вычислительным затратам).