

Деструктори

Трифон Трифонов

Обектно-ориентирано програмиране,
спец. Компютърни науки, 1 поток,
спец. Софтуерно инженерство,
2016/17 г.

23 март 2017 г.

Жизнен цикъл на обект

- За обекта се заделя памет и се свързва с неговото име
- Извиква се подходящ конструктор на обекта
- Работа с обекта (достъп до компоненти на обект, изпълняване на операции)
- Достига се края на областта на действие на обекта
- Извиква се деструкторът на обекта
- Заделената за обекта памет се освобождава

Ролята на деструкторите

- Извършване на заключителни действия

Ролята на деструкторите

- Извършване на заключителни действия
 - затваряне на файл, мрежова или друга връзка

Ролята на деструкторите

- Извършване на заключителни действия
 - затваряне на файл, мрежова или друга връзка
 - освобождаване на заделена памет

Ролята на деструкторите

- Извършване на заключителни действия
 - затваряне на файл, мрежова или друга връзка
 - освобождаване на заделена памет
 - уведомяване за разрушаването на обекта

Ролята на деструкторите

- Извършване на заключителни действия
 - затваряне на файл, мрежова или друга връзка
 - освобождаване на заделена памет
 - уведомяване за разрушаването на обекта
- Деструкторът е противоположен на конструктора

Ролята на деструкторите

- Извършване на заключителни действия
 - затваряне на файл, мрежова или друга връзка
 - освобождаване на заделена памет
 - уведомяване за разрушаването на обекта
- Деструкторът е противоположен на конструктора
- Извиква се автоматично при унищожаване на обекта

Ролята на деструкторите

- Извършване на заключителни действия
 - затваряне на файл, мрежова или друга връзка
 - освобождаване на заделена памет
 - уведомяване за разрушаването на обекта
- Деструкторът е противоположен на конструктора
- Извиква се автоматично при унищожаване на обекта
 - излизане от област на действие

Ролята на деструкторите

- Извършване на заключителни действия
 - затваряне на файл, мрежова или друга връзка
 - освобождаване на заделена памет
 - уведомяване за разрушаването на обекта
- Деструкторът е противоположен на конструктора
- Извиква се автоматично при унищожаване на обекта
 - излизане от област на действие
 - извикване на `delete` или `delete []`

Дефиниране на деструктор

- `<име-на-клас>::~~<име-на-клас>() { <тяло> }`
- Всеки клас може да има **точно един** деструктор
- Ако не бъде дефиниран явно деструктор, **се дефинира системен** с празно тяло
- Ако обектът използва динамична памет, системният деструктор **няма да я освободи**
 - трябва да се дефинира явен деструктор

Управление на динамичната памет

- Заделяме блок динамична памет

Управление на динамичната памет

- Заделяме блок динамична памет
- Подаваме указателя към заделената памет на функция

Управление на динамичната памет

- Заделяме блок динамична памет
- Подаваме указателя към заделената памет на функция
- Копираме указателя в друга променлива

Управление на динамичната памет

- Заделяме блок динамична памет
- Подаваме указателя към заделената памет на функция
- Копираме указателя в друга променлива
- Записваме указателя като член-данна в някакъв обект

Управление на динамичната памет

- Заделяме блок динамична памет
- Подаваме указателя към заделената памет на функция
- Копираме указателя в друга променлива
- Записваме указателя като член-данна в някакъв обект
- **Проблем:** Коя част от програмата “носи отговорност” да освободи динамичната памет, когато вече не е нужна?

Стратегия 1: Собственост

- точно един от указателите се счита за **собственик**

Стратегия 1: Собственост

- точно един от указателите се счита за **собственик**
- другите указатели се считат за **ползватели**

Стратегия 1: Собственост

- точно един от указателите се счита за **собственик**
- другите указатели се считат за **ползватели**
- динамичната памет се **освобожава през собственика**

Стратегия 1: Собственост

- точно един от указателите се счита за **собственик**
- другите указатели се считат за **ползватели**
- динамичната памет се **освобождава през собственика**
- програмата трябва да включва логика, която гарантира, че **ползвателите няма да достъпват паметта след нейното освобождаване**

Стратегия 1: Собственост

- точно един от указателите се счита за **собственик**
- другите указатели се считат за **ползватели**
- динамичната памет се **освобождава през собственика**
- програмата трябва да включва логика, която гарантира, че **ползвателите няма да достъпват паметта след нейното освобождаване**
- **Идея №1:** собственикът е поле на обект и **паметта се освобождава от деструктора**

Стратегия 1: Собственост

- точно един от указателите се счита за **собственик**
- другите указатели се считат за **ползватели**
- динамичната памет се **освобожава през собственика**
- програмата трябва да включва логика, която гарантира, че **ползвателите няма да достъпват паметта след нейното освобождение**
- **Идея №1:** собственикът е поле на обект и **паметта се освобожава от деструктора**
- **Идея №2:** пази се **единствен** указател-собственик към заделената памет, а паметта се достъпва **индиректно чрез селектор на обекта**

Стратегия 1: Собственост

- точно един от указателите се счита за **собственик**
- другите указатели се считат за **ползватели**
- динамичната памет се **освобожава през собственика**
- програмата трябва да включва логика, която гарантира, че **ползвателите няма да достъпват паметта след нейното освобождение**
- **Идея №1:** собственикът е поле на обект и **паметта се освобожава от деструктора**
- **Идея №2:** пази се **единствен** указател-собственик към заделената памет, а паметта се достъпва **индиректно чрез селектор на обекта**
- докато обектът е “жив”, паметта е валидна

Стратегия 1: Собственост

- точно един от указателите се счита за **собственик**
- другите указатели се считат за **ползватели**
- динамичната памет се **освобождава през собственика**
- програмата трябва да включва логика, която гарантира, че **ползвателите няма да достъпват паметта след нейното освобождаване**
- **Идея №1:** собственикът е поле на обект и **паметта се освобождава от деструктора**
- **Идея №2:** пази се **единствен** указател-собственик към заделената памет, а паметта се достъпва **индиректно чрез селектор на обекта**
- докато обектът е “жив”, паметта е валидна
- има конструкции, които позволяват “прехвърляне” на собственост

Стратегия 2: Умни указатели

- споделена собственост между няколко указателя

Стратегия 2: Умни указатели

- споделена собственост между няколко указателя
- поддържа се **общия брой на всички указатели** към дадена заделена памет

Стратегия 2: Умни указатели

- споделена собственост между няколко указателя
- поддържа се **общия брой на всички указатели** към дадена заделена памет
- когато всички указатели-собственици бъдат “изгубени”, паметта се освобождава

Стратегия 2: Умни указатели

- споделена собственост между няколко указателя
- поддържа се **общия брой на всички указатели** към дадена заделена памет
- когато всички указатели-собственици бъдат “изгубени”, паметта се освобождава
- **Идея №1:** умните указатели могат да бъдат **обекти, чиито деструктор сигнализира** за умирането им

Стратегия 2: Умни указатели

- споделена собственост между няколко указателя
- поддържа се **общия брой на всички указатели** към дадена заделена памет
- когато всички указатели-собственици бъдат “изгубени”, паметта се освобождава
- **Идея №1:** умните указатели могат да бъдат **обекти, чиито деструктор сигнализира** за умирането им
- **Идея №2:** понякога са полезни “слаби” указатели-ползватели, като за тях **не се гарантира**, че сочат към валидна памет

Разрушаване на масиви от обекти

- При **заделяне** на масиви от обекти, се извиква конструктор за всеки обект

Разрушаване на масиви от обекти

- При **заделяне** на масиви от обекти, се извиква конструктор за всеки обект
 - ако масивът е заделен в стека, можем да укажем **отделен конструктор за всеки обект**

Разрушаване на масиви от обекти

- При **заделяне** на масиви от обекти, се извиква конструктор за всеки обект
 - ако масивът е заделен в стека, можем да укажем **отделен конструктор за всеки обект**
 - ако масивът е заделен динамично, извиква се **конструкторът по подразбиране**

Разрушаване на масиви от обекти

- При **заделяне** на масиви от обекти, се извиква конструктор за всеки обект
 - ако масивът е заделен в стека, можем да укажем **отделен конструктор за всеки обект**
 - ако масивът е заделен динамично, извиква се **конструкторът по подразбиране**
- При **разрушаване** на масив от обекти, за всеки един от тях трябва да се извика деструктор

Разрушаване на масиви от обекти

- При **заделяне** на масиви от обекти, се извиква конструктор за всеки обект
 - ако масивът е заделен в стека, можем да укажем **отделен конструктор за всеки обект**
 - ако масивът е заделен динамично, извиква се **конструкторът по подразбиране**
- При **разрушаване** на масив от обекти, за всеки един от тях трябва да се извика деструктор
 - при масиви, заделени в стека това става **автоматично**

Разрушаване на масиви от обекти

- При **заделяне** на масиви от обекти, се извиква конструктор за всеки обект
 - ако масивът е заделен в стека, можем да укажем **отделен конструктор за всеки обект**
 - ако масивът е заделен динамично, извиква се **конструкторът по подразбиране**
- При **разрушаване** на масив от обекти, за всеки един от тях трябва да се извика деструктор
 - при масиви, заделени в стека това става **автоматично**
 - при масиви, заделени динамично, **трябва** да бъде използвана операцията **delete []**, а не **delete!**

Разрушаване на масиви от обекти

- При **заделяне** на масиви от обекти, се извиква конструктор за всеки обект
 - ако масивът е заделен в стека, можем да укажем **отделен конструктор за всеки обект**
 - ако масивът е заделен динамично, извиква се **конструкторът по подразбиране**
- При **разрушаване** на масив от обекти, за всеки един от тях трябва да се извика деструктор
 - при масиви, заделени в стека това става **автоматично**
 - при масиви, заделени динамично, **трябва** да бъде използвана операцията `delete []`, а не `delete!`
 - `delete []` се грижи да извика **деструктор на всеки обект** от масива