Table of Contents

Item 17: Store new ed objects in smart pointers in standalone statements.

فرض کنید که ما یک تابع به منظورتشخیص اولویت پردازش داریم و یک تابع دوم نیز داریم که برای پردازش بر روی یک سری Widget که به صورت داینامیک هستند نوشته شده است:

```
int priority();
void processWidget(std::shared_ptr<Widget> pw, int priority);
```

استفاده از شیء برای مدیریت منابع یک انتخاب عقلانی است(آیتم ۱۳)، همانطور که می‌بینید که PrcoessWidget از اشاره‌گر هوشمند(در اینجا shared_ptr) برای مدیریت داینامیک Widget در پردازشش استفاده کرده است.

فرض کنید یک فراخوانی به صورت زیر به processesWidget داشته باشیم.

```
processWidget(new Widget,priority());
```

صبر کنید، این را یک فراخوانی به حساب نیاورید. این کد کامپایل نخواهد شد. سازنده کلاس shared_ptr نمی‌تواند یک اشاره‌گر خام صریح به عنوان ورودی بگیرد، و هیچ تبدیل غیر صریحی از اشاره‌گر خام با عبارت "new Widget" وجود ندارد که بتوانیم به shared_ptr بدهیم. در هر صورت می‌توانیم کد را به صورت زیر بنویسیم و این کد کامپایل خواهد شد:

```
processWidget(std::shared_ptr<Widget>(new Widget),priority());
```

ممکن است تعجب کنید اگر بفهمید با وجود استفاده از شیء برای مدیریت منابع، این فراخوانی ممکن است نشت منبع داشته باشد.

قبل از این که کامپایلر بتواند یک فراخوانی به processWidget داشته باشد، باید آرگومان‌هایی که به عنوان پارامتر به تابع فرستاده می‌شوند را ارزیابی کند. آرگومان دوم یک فراخوانی به اولویت تابع می‌باشد و مشکلی با آن نداریم، اما آرگومان اول یعنی std::shared_ptr<Widget>(new Widget) از دو قسمت تشکیل شده است.

- اجرای عبارت new Widget
- فراخوانی سازنده کلاس shared_ptr

کامپایلر C++ به ما یک تضمین در مورد آزادی عمل وسیع، در مورد مشخص کردن ترتیب پارامترهای ورودی میدهد.(این خیلی متفاوت از زبان‌هایی مثل C# و Java هست که پارامترهای توابع همیشه باید در یک ترتیب خاصی ارزیابی شوند.) عبارت new Widget باید قبل از سازنده‌ی shared_ptr اجرا شود،

چرا که نتیجه‌ی عبارت هست که بـه عنـوان ورودی سـازنده‌ی کلاس shared_ptr مـورد اسـتفاده قـرار میگیرد، اما فراخوانی به تابع priority می‌تواند اول، دوم، و یا سوم باشد. اگر کامپـایلر انتخـاب کنـد کـه انتخاب تابع priority دوم باشد(شاید کامپایلر به خاطر این که بتواند یک کد بهتر تولید کند این کـار را انجام دهد)، در این صورت مراحل اجرای کد به صورت زیر درخواهد آمد.

۱. اجرای new Widget
۲. فراخوانی تابع priority
۳. فراخوانی سازنده کلاس shared_ptr

حال فرض کنید که ما در هنگام فراخوانی تابع priority به یک exception برخورد کنیم. در این مـورد، اشاره‌گری که از new Widget برگشت داده شده، از دست خواهد رفت، چرا که کـد نتوانسـته آن را در shared_ptr ذخـیره کنـد، و shared_ptr همـان چـیزی بـود کـه بـرای جلوگـیری از نشـت منبـع مـا می‌خواستیم از آن استفاده کنیم. بنابراین در هنگام فراخوانی تابع processWidget مـا می‌تـوانیم نشـت حافظـه داشـته باشـیم، و این مـورد بـه خـاطر وجـود exception فی مـا بین سـاختن یـک منبع (new Widget) و تحویل دادن منبع به یک کلاس مدیریت منبع رخ داده است.

راهی که برای جلوگیری از چنین مسایلی وجود دارد، خیلی ساده است: استفاده از یک عبارت جدا بـرای ساختن Widget و ذخیره‌ی آن در یک اشاره‌گر هوشمند، سپس پاس دادن اشـاره‌گر هوشـمند بـه تـابع processWidget :

```
shared_ptr<Widget> pw(new Widget);    //store newed object in a smart
                                      //pointer in a standalone statement

processWidget(pw,priority());         //this call won't leak
```

این کد بدون مشکل کار می‌کند چون کامپایلر نمی‌تواند ترتیب اجرای عبارت‌هـای جـدا از هم را بـه هم بزند. در این کد عبارت new Widget و فراخوانی به shared_ptr در یک عبارت جداگانه قـرار دارنـد و تابع priority در یک عبارت جدا، بنابراین کامپایلر اجازه ندارد که قبـل از اجـرای آن دو کـه در عبـارت قبلی قرار دارند، تابع priority را صدا بزند.