Table of Contents

Item 16: Use the same form in corresponding uses of new and delete .

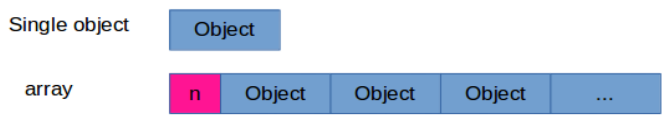
اشکال برنامه‌ی زیر چیست؟

```
std::string *stringArray = new std::string[100];
....
delete stringArray;
```

همه چیز به نظر طبق همان ترتیبی است که باید باشد. new با delete تطبیق دارد. ولی، یک چیز کاملا اشتباه است. کاری که برنامه می‌کند در این مورد نامشخص است. در واقع، ۹۹ تا از ۱۰۰ شیء مربوط به stringArray احتمالا به درستی حذف نخواهند شد، چرا که destructor شان هرگز فراخوانی نخواهد شد.

وقتی که از کلمه‌ی کلیدی new برای تولید داینامیک یک شیء استفاده می‌کنید، دو اتفاق می‌افتد. یک، حافظه اختصاص داده می‌شود(با استفاده از یک تابعی که اپراتور new فراخوانی می‌کند- آیتم ۴۹ و ۵۱ را ببینید). دوم، یک یا چندین سازنده برای آن معماری صدا زده می‌شود. وقتی که از عبارت delete استفاده می‌کنید، دو اتفاق دیگر می‌افتد: یک یا چندین مخرب برای حافظه صدا زده می‌شود، سپس حافظه deallocate می‌شود(با استفاده از تابعی که اپراتور delete نامیده می‌شود-آیتم ۵۱ را ببینید). سوال بزرگی که برای delete پیش میآید این است که: چه تعداد از اشیایی که در حافظه جا گرفته‌اند حذف می‌شوند؟ پاسخی که به این سوال می‌دهیم مشخص کننده تعدادی مخرب‌هایی است که باید صدا زده شود.

در واقع، سوال ساده‌تر از این چیزی هست که به نظر میرسد: در واقع اشاره‌گری که حذف می‌شود آیا اشاره به یک شیء دارد و یا به آرایه‌ای از اشیاء؟ این یک سوال حیاتی است، چرا که طرحبندی حافظه برای یک تک شیء به صورت کلی متفاوت از طرحبندی حافظه برای آرایه‌هاست. به طور مشخص، حافظه‌ای که برای یک آرایه گرفته می‌شود معمولا سایز آرایه نیز در آن گنجانده می‌شود، بنابراین کار را برای حذف کردن آسان‌تر می‌کند، چون وقتی تعداد آرایه مشخص است، می‌دانیم چند بار باید مخرب صدا زده شود. حافظه‌ای که برای یک تک شیء گرفته می‌شود همچین اطلاعاتی را ندارد. در واقع شما می‌توانید این تفاوت در طرحبندی یا layout را به صورت زیر ببینید.



البته این شکل فقط برای مثال زدن بود، و کامپایلرها مجبور نیستند که حتما به همین شکل این مورد رو پیاده‌سازی کنند اگر چه خیلی‌هاشون همین کار رو می‌کنند.

وقتی شما از delete روی یک اشاره‌گر استفاده می‌کنید، تنها راهی که delete می‌تواند اطلاعاتی در مورد size آرایه داشته باشد این هست که این اطلاعات را خودتان به delete بدهید. اگر از **براکت** در هنگام استفاده از delete ، استفاده کنید، delete فرض می‌کند که به یک آرایه اشاره کرده است. در غیر این صورت، delete فرض می‌کند که روی یک تک شیء فراخوانی شده است:

```
std::string *stringPtr1=new std::string;
std::string *stringPtr2=new std::string[100];
....
delete stringPtr1;  //delete an object
delete [] stringPtr2; //delete an array of objects
```

حال چه اتفاقی می‌افتد اگر از [] برای stringPtr1 استفاده کنیم؟ نتیجه نامشخص خواهد بود. با فرض طرح بندی بالا، delete شروع به خواندن حافظه می‌کند و array size را با تفسیر آن به دست می‌آورد، و شروع به invoke کردن به اندازه‌ی array size می‌کند، delete این کار را بدون توجه به این حقیقت که آن قسمت از حافظه جزو آرایه نیست، انجام میدهد، و ممکن است بدون این که شیءای داشته باشد مشغول فراخوانی destructor باشد.

حال چه اتفاقی می‌افتد اگر از [] برای stringPtr2 استفاده نکنیم؟ این کار نیز نتیجه‌ی نامشخصی خواهد داشت، ولی می‌توانید ببینید که چطور برخی از destructor ها فراخوانی نمی‌شوند. به علاوه، در مورد متغیرهای built-in مانند int که destructor ندارند این کار ممکن است نامشخص و یا حتی مضر نیز باشد.

قانونی که برای حذف حافظه‌ی داینامیک داریم ساده است: اگر در عبارت new از [] استفاده کردید، باید در عبارت delete متناظر از [] استفاده کنید، و اگر در عبارت new از [] استفاده نکردید در عبارت متناظر delete نیز این کار را انجام ندهید.

این قانون مشخصا در مورد نوشتن کلاسی که یک اشاره‌گر به حافظه داینامیک دارد و چندین سازنده نیز دارد، مهم است، چون شما باید در همه‌ی سازنده‌ها به یک فرمت یکسان از new استفاده کنید. اگر این کار را انجام ندهید، چطور می‌تونید بفهمید که از کدام فرمت delete باید در مخرب استفاده کرد؟

این قانون همچنین در مورد typedef ها نیز مهم هست، چون کسی که typedef را می‌نویسد، باید در اسناد حتما اشاره کند که چه delete ای باید استفاده شود. به طور مثال:

```
typedef std::string addressLines[4]; //a person's address has 4 lines, each of which is a string
```

چون addressLines یک آرایه است، استفاده از new برای آن:

```
std::string *pa1=new addressLines; //note that "new AddressLines" returns a string*,
                                   //just like "new string[4]" would
```

باید delete هم به صورت array برای آن تعریف شود.

```
delete [] pa1;
```

برای جلوگیری از چنین مشکلی، از typedef برای آرایه‌ها استفاده نکنید. از اونجایی که کتابخانه‌ی استاندارد C++ شامل string و vector نیز هست، و این template ها نیاز ما برای تخصیص حافظه به صورت داینامیک را تقریبا به صفر کاهش داده است، پس این که از typedef برای آرایه‌ها استفاده نکنیم ساده و بدون مشکل خواهد بود. به طور مثال، در مورد AddressLines می‌توانیم به صورت vector<string> استفاده کنیم.