خلاصه‌ی برخی از موضوعات مطرح شده در درس طراحی کامپایلر

خلاصه‌ای از موضوعات مطرح شده در جلسه‌های پایانی درس در این مستند بیان می‌گردد.

جمع‌آوری زباله

در دسته‌ی تخصیص حافظه زمان اجرا، شیوه‌ی مدیریت پشته را بررسی کرده‌ایم. در ادامه به مدیریت حافظه‌ای که از روی Heap تخصیص داده می‌شود، می‌پردازیم. برای مثال، از شبه کد زیر که در زبان Python نوشته شده است، استفاده می‌شود. در این شبه کد کلاس Test یک فیلد به نام var دارد. برای خوانایی بیشتر مثال‌ها، به سازنده‌ی هر شیء از این کلاس یک پارامتر به عنوان نام فرستاده می‌شود و با استفاده از این نام اشیاء متمایز می‌گردند.

```
class Test(object):
      def __init(self, name):
            pass
      var = None

def func():
      c = Test('C')

a = Test('A')
b = Test('B')
a.var = Test('D')
a.var.var = b
func()
```

خطاهای مدیریت حافظه‌ی Heap

ممکن است حافظه‌ی اختصاص داده شده به برخی از این اشیاء که دیگر نیازی به آنها نیست آزاد نگردد (Memory leaks)، یا ممکن است به یک شیء دسترسی انجام شود که هنوز حافظه‌ای برای آن تخصیص داده نشده یا قبلا آزاد شده است (Dangling pointer) یا گاهی ممکن است حافظه‌ی اختصاص داده شده به یک شیء دو بار آزاد گردد (Double free). برای جلوگیری از این خطاها، بسیاری از زبان‌های برنامه‌نویسی سطح بالا مدیریت حافظه‌ی Heap را به صورت خودکار و در قالب Garbage Collection انجام می‌دهند.

مجموعه‌ی ریشه (Root Set)

مجموعه‌ای از اشیاء که متغیرهای محلی، سراسری و ایستا به صورت مستقیم به آنها ارجاع می‌دهند. در مثال، اشیائی که توسط متغیرهای سراسری a و b و متغیر محلی c (در هنگام اجرای تابع func) به آنها اشاره می‌شود، مجموعه‌ی ریشه محسوب می‌شوند. بنابراین مجموعه‌ی ریشه شامل A، B و C هستند. دقت کنید که پس از پایان فراخوانی تابع func، C مجموعه‌ی ریشه محسوب نمی‌شود.

زباله (Garbage)

زباله در یک برنامه‌ی در حال اجرا، به قسمتی از حافظه گفته می‌شود که در ادامه‌ی برنامه از آن استفاده نمی‌شود. هدف جمع‌آوری خودکار زباله (Garbage Collection)، یافتن و آزاد کردن زباله‌ها است.

گراف ارجاع (Reference Graph)

گراف ارجاع، گرافی است که رأس‌های آن اشیاء موجود در حافظه هستند و یک یال جهت‌دار بین دو شیء وجود یک ارجاع از شیء اول به شیء دوم را نشان می‌دهد. در این گراف، مجموعه‌ی ریشه به شکلی مشخص می‌گردند. زباله‌ها در این گراف، اشیائی هستند از مجموعه‌ی ریشه نمی‌توان به آنها رسید.

ایجاد زباله

در مثال، پس از فراخوانی تابع func شیء ساخته شده در این تابع قابل دسترسی نیست و زباله محسوب می‌گردد. اگر خط زیر به پایان مثال اضافه شود، اشیاء A و B هم زباله محسوب می‌شوند اما B و E زباله نیستند چون متغیرهای b و a به آنها اشاره می‌کنند.

```
a = Test('E')
```

جمع‌آوری زباله به روش شمارش ارجاع (Reference Counting)

در این روش، به ازای هر شیء تعداد ارجاعات انجام شده به آنها نگهداری می‌شود؛ این مقدار در هنگام انتساب به متغیرها و فیلدها و ورود به و خروج از توابع به روز می‌شود. زباله‌ها اشیائی هستند که تعداد ارجاعات آنها صفر است. با آزاد شدن این زباله‌ها و از بین رفتن ارجاعات آنها، زباله‌های بیشتری تشخیص داده می‌شوند.

مشکلات روش شمارش ارجاع

الف) حلقه‌های ارجاع به عنوان زباله تشخیص داده نمی‌شوند (چون تعداد ارجاع به آنها بیشتر از یک باقی می‌ماند حتی اگر در گراف ارجاع قابل دسترسی نباشند). ب) به ازای هر شیء باید تعداد ارجاعات نگهداری شود (سربار حافظه). ج) برای دسترسی همزمان به تعداد ارجاعات با وجود بندها باید همگام‌سازی انجام شود (با قفل‌ها). د) تشخیص زباله گاهی موجب

وقفه‌ی غیر قابل پیش‌بینی می‌شود (گاهی امکان دارد تعداد زیادی شیء در یک لحظه به عنوان زباله تشخیص داده شوند) و این تأخیر برای کاربردهای بلادرنگ (Realtime) مناسب نیست.

جمع‌آوری زباله مبتنی بر Tracing

در این روش‌ها، گراف ارجاع با شروع از مجموعه‌ی ریشه پیمایش می‌شود و اشیائی که قابل دسترسی هستند علامت زده می‌شوند. سپس، اشیائی که علامت زده نشده‌اند به عنوان زباله تشخیص داده می‌شوند. در این دسته از روش‌ها، حلقه‌ها به درستی زباله تشخیص داده می‌شوند.

ارزیابی الگوریتم‌های جمع‌آوری زباله

الگوریتم‌های جمع‌آوری زباله را با توجه به متغیرهایی مثل موارد زیر ارزیابی می‌کنند. الف) سرعت الگوریتم. ب) سربار حافظه. ج) وقفه‌های ایجاد شده در برنامه به علت فعال شدن جمع‌آوری زباله. د) ایجاد محلی‌گرایی مکانی (اشیائی که با هم ایجاد می‌شوند در کنار هم در حافظه قرار گیرند).

تولید کد نهایی (کد ماشین)

در فاز آخر کامپایلر کد میانی به کد ماشین تبدیل می‌شود. دقت کنید که این گام پس از تخصیص رجیسترها انجام می‌شود. بنابراین، مشخص شده است که چه عملوندهای کد میانی در یک رجیستر یا در قسمتی از حافظه قرار می‌گیرند.

مجموعه‌ی دستورات ماشین

مجموعه‌ی دستورات یک ماشین در Instruction Set Architecture (ISA) توسط شرکت سازنده‌ی پردازنده مشخص می‌گردد. بیشتر دستورات کد میانی به سادگی به یک یا چند دستور ماشین تبدیل می‌شوند. برای در مثال زیر کد میانی به دو شکل به کد ماشین تبدیل شده است (با فرض مجموعه دستوراتی که در کلاس مطرح شد).

کد میانی	کد نهایی الف	کد نهایی ب

`a = a + 5`	`mov r1, 5`
	`mov r2, [a]`
	`add r2, r1`
	`mov [a], r2`
		`add [a], 5`

تفاوت معماری‌ها

معماری پردازنده‌ها را می‌توان در دو دسته‌ی CISC و RISC دسته‌بندی کرد (البته بسیاری از پردازنده‌ها برخی از ویژگی‌های هر دو دسته را به ارث برده‌اند). در مقایسه با پردازنده‌های RISC، پردازنده‌های CISC ویژگی‌هایی دارند که برخی از گام‌های

تولید کد نهایی و تخصیص رجیستر را دشوارتر می‌کند، از جمله: الف) تعداد رجیسترهای آنها کمتر است. ب) رجیسترها به دسته‌های متفاوتی تقسیم می‌شوند و برخی از دستورات فقط برای برخی از این رجیسترها قابل انجام است. ج) حالت‌های متفاوتی برای آدرس‌دهی دستورات وجود دارد (برای نمونه عمل جمع می‌تواند روی دو رجیستر، روی یک رجیستر و یک خانه از حافظه، روی یک رجیستر و یک عدد ثابت یا روی یک خانه‌ی حافظه و یک عدد ثابت اجرا شود). د) برای عملگرهای Binary (با دو عملوند) دستوراتی وجود دارد که مقصد دستور یکی از علوندها است (مثل add r1, r2 که حاصل جمع r1 و r2 را به r1 می‌ریزد). ه) برخی از دستورات مقدار برخی از رجیسترهای غیر مؤثر در دستور را تغییر می‌دهند.

انتخاب دستورات

ممکن است برای کد میانی دنباله‌ی متفاوتی از دستورات را بتوان تولید کرد. برخی از کامپایلرها الگوریتمی را که برای انتخاب دستورات (گام Instruction Selection) اجرا می‌کنند. در انتخاب دستورات معمولا کد میانی به صورت نمایش گراف DAG (Directed Acyclic Graph) نمایش داده می‌شوند. سپس سعی می‌شود این گراف با استفاده از الگوهایی که از دستورات پردازنده ایجاد می‌شود پوشانده شود (به این کار اصطلاحا Tiling یا کاشی کاری گفته می‌شود). برای نمونه به شکل صفحه‌های ۱۹۲ و ۱۹۳ کتاب Appel مراجعه کنید.

الگوریتم انتخاب دستورات

الگوریتم‌های متفاوتی برای کاشی کاری در انتخاب دستور وجود دارد. در الگوریتم بزرگ‌ترین لقمه‌ی (Maximal Munch)، انتخاب دستورات ریشه شروع می‌شود و همیشه سعی می‌شود از بزرگ‌ترین الگوی ممکن استفاده شود. در الگوریتم مبتنی بر برنامه‌ریزی پویا سعی می‌شود با استفاده از برنامه‌ریزی پویا بهترین کاشی کاری انتخاب شود.

سایر نکته‌ها

سرریز پشته (Stack Overflow)

سیستم عامل به پشته‌ی هر بند اندازه‌ی محدودی اختصاص می‌دهد (این مقدار معمولا قابل تنظیم است). در صورتی که بندی حافظه‌ی پشته را پر کند سرریز پشته رخ می‌دهد و برنامه Crash می‌کند.