اصول طراحی کامپایلرها

حسین کارشناس

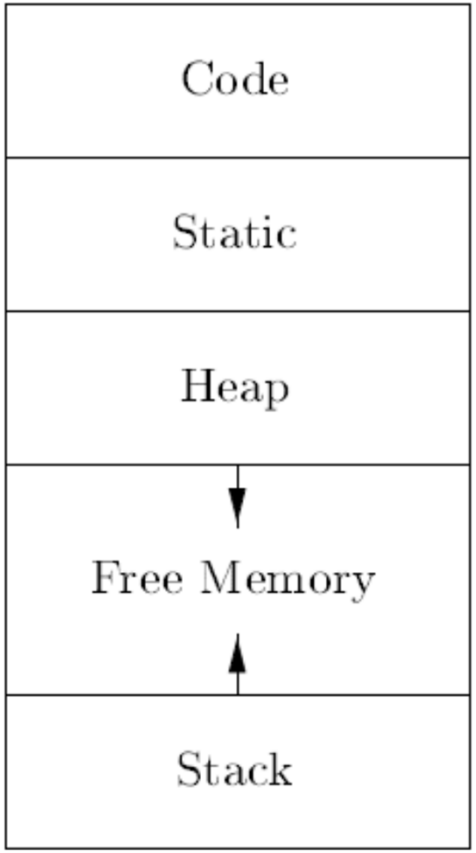
دانشکده مهندسی کامپیوتر

ترم اول ۹۸ – ۹۷

طرح‌بندی حافظه (memory layout)

طرح‌بندی حافظه

- طرح‌بندی حافظه زمان اجرای برنامه‌ها



طرح‌بندی حافظه

- حافظه زمان اجرا توسط سیستم‌عامل مدیریت می‌شود
- تعیین محتوای بخش‌های مختلف در زمان تولید برنامه (کامپایل)
  - بحش کد حاوی کد اجرایی برنامه است که توسط کامپایلر تولید می‌شود
  - بخش داده‌های ایستا حاوی متغیرها و مقادیر مورد استفاده در برنامه
    - در زمان کامپایل اندازه مورد نیاز و آدرس نسبی آنها محاسبه می‌شود
  - پشته حاوی اطلاعات مورد نیاز برای فراخوانی توابع در حین اجرای برنامه
    - اطلاعات قرار گرفته در پشته برای هر فراخوانی در زمان کامپایل مشخص می‌شود
  - Heap برای نگهداری داده‌ها با طول عمر (و اندازه) دلخواه در زمان اجرای برنامه استفاده می‌شود
    - معمولاً اندازه این حافظه هم در زمان کامپایل مشخص می‌شود

طرح‌بندی حافظه داده‌ها

- نیاز به تعیین مکان ذخیره‌سازی مقادیر متغیرها
  - این اطلاعات در مراحل بعدی کامپایل (و اجرا) مورد استفاده قرار می‌گیرد
    - دسترسی به متغیرها بر اساس مکان ذخیره‌سازی آنها در کد نهایی
- تعیین آدرس نسبی (offset) هر متغیر
  - آدرس آن متغیر نسبت به ابتدای یک بخش داده‌ای در حافظه را نشان می‌دهد
  - نگهداری آدرس نسبی متغیرها در جدول نشانه‌ها
  - استفاده از حجم حافظه مورد نیاز برای متغیرها جهت تعیین آدرس نسبی
- حافظه مورد نیاز برای هر متغیر توسط نوع آن متغیر تعیین می‌شود
  - در زمان تعریف آن متغیر مشخص می‌شود

طرح‌بندی حافظه داده‌ها

- عرض (width) یک نوع
  - تعداد واحدهای حافظه مورد نیاز برای ذخیره‌سازی نمونه‌هایی از آن نوع
  - در نظر گرفتن یک عرض پیش‌فرض برای نوع‌های پایه‌ای
    - مثال: عرض نوع int ۴ واحد حافظه و عرض نوع float ۸ واحد حافظه
  - عرض نوع‌های جمعی (aggregate types) برابر مجموع عرض فیلدهای آن
    - مثال: عرض نوع جمعی زیر برابر ۱۲ واحد حافظه است

```
record {  int x;
          float y;  } z;
```

  - عرض نوع آرایه بر اساس نوع درایه‌ها و تعداد آنها تعیین می‌شود
    - مثال: عرض نوع int[2][3] برابر ۲۴ واحد حافظه است

طرح‌بندی حافظه داده‌ها

- تعیین حافظه مورد نیاز برای متغیرها در دستورات تعریف متغیر
  - شناسایی بخش‌هایی از برنامه که امکان تعریف متغیر وجود دارد
    - در گرامر زبان مشخص است
  - مثال: یک گرامر نمونه برای تعریف متغیرها

$$
\begin{aligned}
P &\rightarrow B\ P \mid \epsilon \\
B &\rightarrow \mathbf{\{}\ D\ S\ \mathbf{\}} \\
D &\rightarrow T\ \mathbf{id}\ \mathbf{;}\ D \mid \epsilon \\
T &\rightarrow C\ A \mid \mathbf{record}\ \mathbf{\{}\ D\ \mathbf{\}} \\
C &\rightarrow \mathbf{int} \mid \mathbf{float} \\
A &\rightarrow \mathbf{[}\ \mathbf{num}\ \mathbf{]}\ A \mid \epsilon \\
S &\rightarrow B\ S \mid \epsilon
\end{aligned}
$$

طرح‌بندی حافظه داده‌ها

- محاسبه آدرس نسبی (offset) متغیرها
  - آدرس نسبی یک متغیر با عرض دلخواه، آدرس اولین بایت آن است
  - آدرس نسبت به ابتدای یک بخش داده‌ای محاسبه می‌شود
    - در ابتدای هر بخش داده‌ای offset = 0 است
  - آدرس نسبی هر متغیر برابر با مقدار offset در هنگام تعریف آن متغیر است
  - افزایش مقدار offset به اندازه عرض نوع با مشاهده هر دستور تعریف متغیر

$$\text{T x;} \quad \Rightarrow \quad \text{offset} = \text{offset} + \text{width(T)}$$

طرح‌بندی حافظه داده‌ها

- آدرس نسبی فیلدها در نوع‌های جمعی
  - آدرس نسبی فیلدها نسبت به ابتدای نوع جمعی تعیین می‌شود

- مثال: آدرس هر یک از فیلدها و متغیرها در برنامه زیر

```
float x;
record { float x; float y; } p;
record { int tag; float x; float y; } q;
x = p.x+q.x;
```

- به سه offset متفاوت نیاز است

طرح‌بندی حافظه داده‌ها

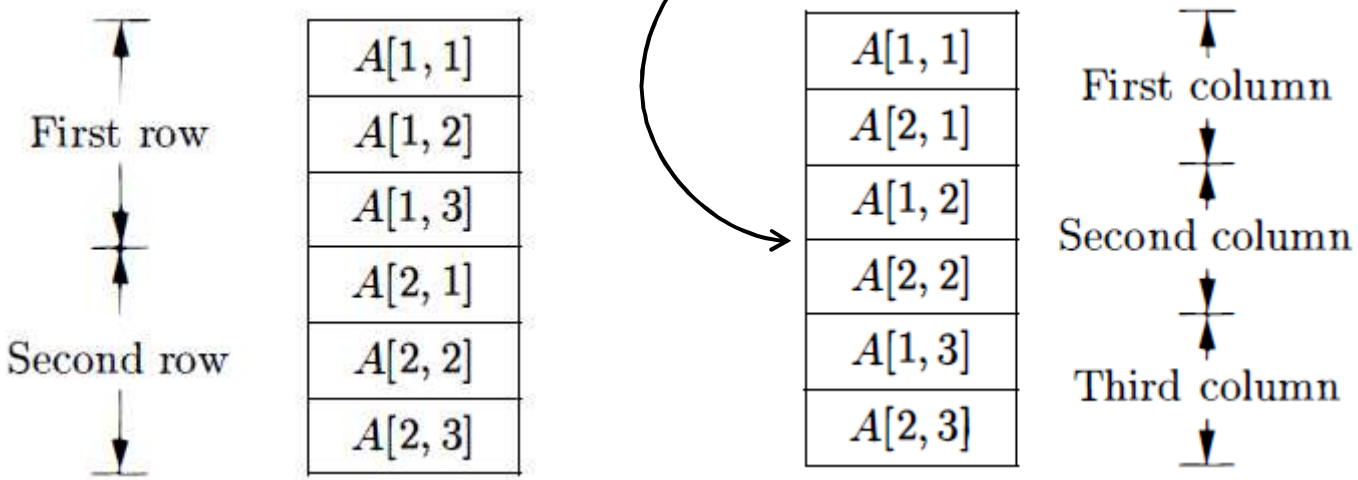
- آدرس نسبی درایه‌های آرایه $x[i_1][i_2]\ldots[i_k]$
  - حافظه اختصاص داده شده به آرایه پیوسته است
  - آدرس نسبی iامین عنصر در آرایه تک بعدی: $base + i \times w$
    - در آرایه k بعدی: $base + i_1 \times w_1 + \ldots + i_k \times w_k$
  - محاسبه آدرس نسبی بر اساس تعداد عناصر ابعاد مختلف آرایه‌ها (n_i):

$$base + i_1 \times n_2 \times \ldots \times n_k \times w + \ldots + i_{k-1} \times n_k \times w + i_k \times w$$

  - اولین اندیس آرایه می‌تواند غیر صفر باشد
    - آدرس iامین عنصر $base + (i - low) \times w$
    - محاسبه $base - low \times w$ بصورت جداگانه در زمان تعریف آرایه
  - در حالت کلی اندیس آرایه می‌تواند در محدوده $low \le i \le high$ باشد

طرح‌بندی حافظه داده‌ها

- طرح‌بندی‌های متفاوت ذخیره‌سازی آرایه در حافظه
  - سطری (row-major)
  - ستونی (column-major)



یک آرایه 3 × 2

- جابجایی ارزش اندیس‌ها در محاسبه آدرس نسبی درایه‌های آرایه

طرح‌بندی حافظه پشته

- نمایش فراخوانی‌ها بصورت درخت فعال‌سازی (activation tree)
  - امکان نمایش تودرتوی طول عمر توابع در زمان اجرای برنامه
  - مثال: درخت فعال‌سازی برای برنامه روبرو

```
Class Main {
   g() : Int { 1 };
   f():  Int { g() };
   main(): Int {{ g(); f(); }};
}
```

  - سوال: درخت فعال‌سازی برنامه روبرو؟

```
Class Main {
   g() : Int { 1 };
   f(x:Int):  Int { if x = 0 then g() else f(x - 1) fi};
   main(): Int {{f(3); }};
}
```

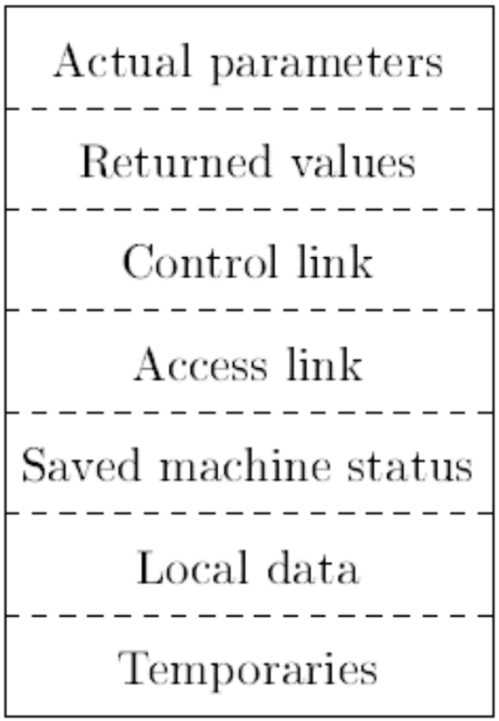
طرح‌بندی حافظه پشته

- تعیین ترتیب فراخوانی‌ها با توجه به درخت فعال‌سازی
  - پیمایش پیش‌ترتیب درخت فعال‌سازی متناظر با دنباله فراخوانی توابع است
  - پیمایش پس‌ترتیب درخت فعال‌سازی متناظر با دنباله بازگشت‌ها از توابع است

- بکارگیری پشته برای پیمایش درخت فعال‌سازی
  - پشته می‌تواند به درستی تابع فعال در زمان فعلی را نشان دهد
  - نگهداری اطلاعات لازم برای مدیریت فعال‌سازی در هنگام فراخوانی توابع
    - رکورد فعال‌سازی (activation record - AR) یا قاب (frame)
    - اطلاعات لازم و ساختار آن باید در زمان کامپایل توسط کامپایلر مشخص شود

طرح‌بندی حافظه پشته

- رکورد فعال‌سازی
  - حاوی اطلاعات مورد نیاز در طول عمر یک تابع

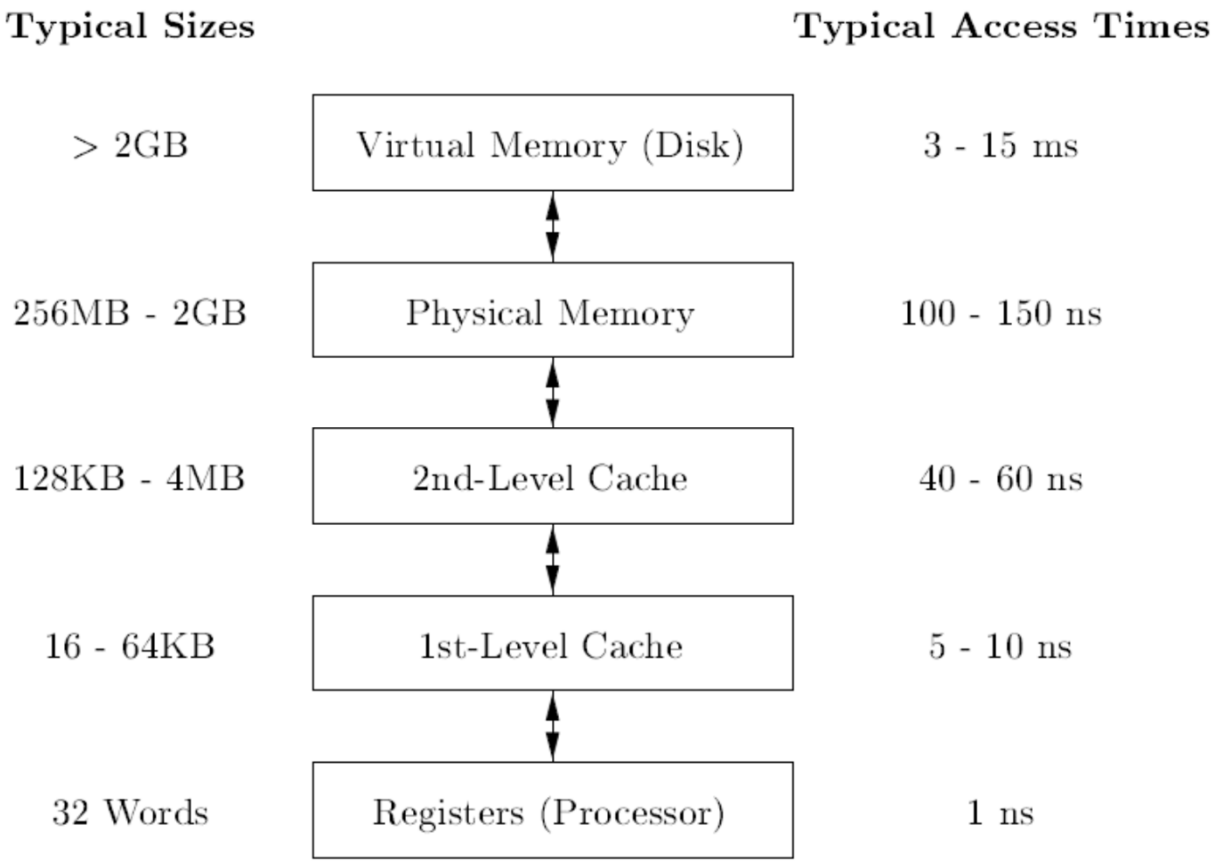


طرح‌بندی حافظه Heap

- برای پشتیبانی از تخصیص پویای حافظه در زمان اجرا برنامه
  - امکان تعیین حافظه مورد نیاز برای تخصیص پویا در زمان کامپایل
    - مثال: حافظه مورد نیاز برای یک شیء (object)
- استفاده از مدیریت حافظه برای تخصیص و آزادسازی فضا
  - بکارگیری کارآمد فضا
    - کاهش چندپارگی (fragmentation) در تخصیص فضا
  - کارایی بالا در اجرای برنامه
    - کاهش سربار مدیریت حافظه، لحاظ ویژگی محلی بودن (locality) در تخصیص حافظه
  - ...

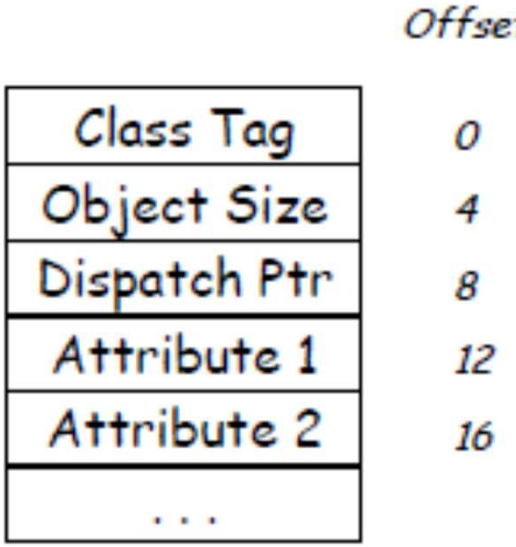
طرح‌بندی حافظه Heap

- بکارگیری سلسله مراتب حافظه‌ها در تخصیص حافظه



طرح‌بندی حافظه Heap

- طرح‌بندی حافظه اشیاء
  - با توجه به تعریف نوع (کلاس) مربوطه تعیین می‌شود
  - با توجه به ویژگی‌های در نظر گرفته شده در زبان‌های مختلف فرق می‌کند
  - عرض و آدرس نسبی هر یک از اجزای نوع در زمان کامپایل تعیین می‌شود



طرح‌بندی حافظه Heap

- پشتیبانی از وراثت
  - آدرس نسبی و عرض فیلدها در تمام اشیاء ثابت است
  - فیلدهای زیرنوع‌ها پس از فیلدهای ابرنوع‌ها قرار می‌گیرد
    - رویکرد افزایشی در افزودن ویژگی‌های جدید به زیرنوع‌ها

$$A_n < \ldots < A_3 < A_2 < A_1$$

طرح‌بندی حافظه Heap

- مثال: طرح‌بندی حافظه برای اشیایی با رابطه وراثت

```
Class A {
    a: Int <- 0;
    d: Int <- 1;
    f(): Int { a <- a + d };
};
```

```
Class B inherits A {
    b: Int <- 2;
    f(): Int { a };
    g(): Int { a <- a - b };
};
```

```
Class C inherits A {
    c: Int <- 3;
    h(): Int { a <- a * c };
};
```

Offset Class	0	4	8	12	16	20
A	Atag	5	*	a	d	
B	Btag	6	*	a	d	b
C	Ctag	6	*	a	d	c

طرح‌بندی حافظه Heap

- جدول توابع (dispatch)
  - دارای آدرس هر یک از توابع تعریف شده در یک کلاس
    - این آدرس‌ها در زمان کامپایل مشخص می‌شود
  - مثال:

```
Class A {
    a: Int <- 0;
    d: Int <- 1;
    f(): Int { a <- a + d };
};
Class  B inherits A {
    b: Int <- 2;
    f(): Int { a };
    g(): Int { a <- a - b };
};
Class  C inherits A {
    c: Int <- 3;
    h(): Int { a <- a * c };
};
```

Offset Class	0	4
A	fA	
B	fB	g
C	fA	h