# برنامه‌سازی پیشرفته

تمرین شماره‌ی شش

---

مدرس: رامتین خسروی، امین صادقی
طراحان: احسان حاج‌یاسینی، سپهر سامنی، فراز یزدانی
موعد تحویل: دوشنبه ۴ اردیبهشت ۱۳۹۶

## مقدمه

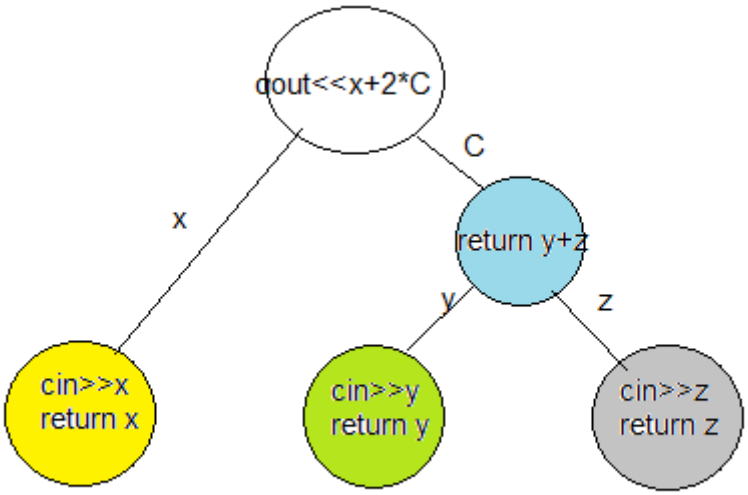
همان‌طور که می‌دانید، یک کامپیوتر با تنها یک پردازنده می‌تواند در آن واحد بیشتر از یک کار را انجام دهد؛ برای مثال شما در موبایل خود - که یک کامپیوتر با یک پردازنده است - می‌توانید همان زمانی که موسیقی پخش می‌کنید در گوگل سرچ کنید. اما چطور ممکن است؟ چگونه پردازنده‌ای با یک هسته‌ی پردازشی که در آن واحد فقط می‌تواند یک دستور اسمبلی را اجرا کند، می‌توانند همزمان چندین برنامه را اجرا کنند؟ در حقیقت کامپیوتر شما در آن واحد فقط یک کار را انجام می‌دهد، اما به شما این حس را القا می‌کند که در حال انجام کارهای گوناگونی است. برای مثال، فرض کنید شما می‌خواهید دو کار f1 و f2 را انجام دهید. روند کار به این صورت است که کامپیوتر در ۱۰۰ میلی‌ثانیه قسمتی از کار f1 را انجام می‌دهد، سپس در ۱۰۰ میلی‌ثانیه‌ی بعدی قسمتی از کار f2 را انجام می‌دهد، سپس دوباره برای ۱۰۰ میلی ثانیه به f1 بازمی‌گردد و این چرخه تا پایان هر دو برنامه ادامه پیدا می‌کند. با تغییرات اندک در همین روش، نسخه‌های مختلفی از روش‌های «چندکارگی» ایجاد می‌شود که در کامپیوترها استفاده می‌شود. در ادامه با بعضی از روش‌های چندکارگی آشنا خواهید شد. در این تمرین شما باید این عملکرد کامپیوتر را شبیه‌سازی کنید. در ادامه مفاهیم مختلفی را که در حوزه‌ی چندکارگی استفاده می‌شود بررسی می‌کنیم.

## تعاریف

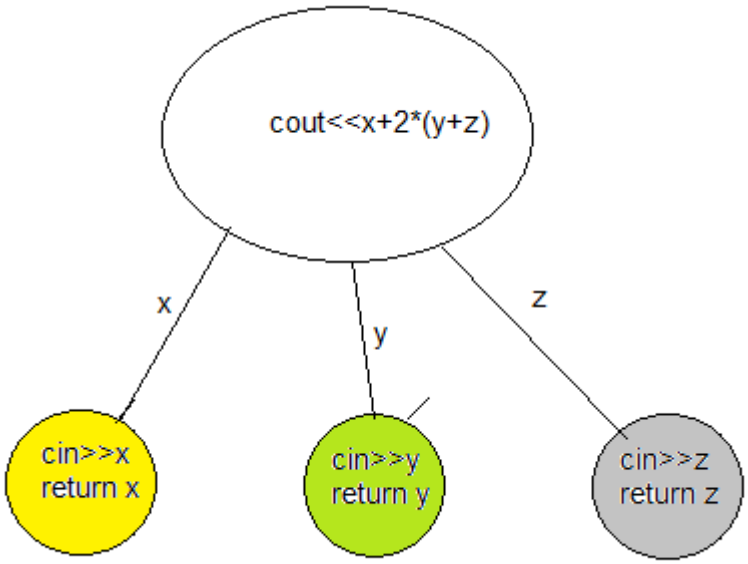
توجه: دقت کنید که این تمرین، یک مدل بسیار ساده و در برخی موارد نادقیق از روش‌های امروزیِ استفاده‌شده برای این هدف است. شما در آینده، در درس سیستم عامل، با جزئیات این روندها آشنا خواهید شد؛ لذا تعریف‌های زیر را صرفاً درقالب این تمرین معتبر بدانید. :)

### برنامه (Functor)

شما در درس ریاضیات گسسته با گراف و خواص آن آشنا شده‌اید. یکی از کاربردهای گراف در علوم کامپیوتر، نمایش یک برنامه‌ی کامپیوتری است. به‌طور مثال فرض کنید برنامه ای دارید که سه مقدار $x$، $y$ و $z$ را از کاربر می‌گیرد و عبارت $x + 2 * (y + z)$ را چاپ می‌کند. این برنامه را می‌توان به صورت یک گراف (درخت) به شکل زیر نمایش داد.




هر گره این گراف شامل قطعه‌کدهایی است که برای اجرا به گره‌های فرزند آن گره وابسته‌اند و هر گره خروجی خود را با یالی به گره پدر خود می‌دهد. برای مثال، در گراف بالا برای اجرای برنامه و محاسبه‌ی نتیجه‌ی نهایی (گره سفید) باید ابتدا گره‌های آبی و زرد اجرا شوند. خود گره‌های آبی و زرد هم یک برنامه (فانکتور) هستند. توجه داشته باشید که یک برنامه را به صورت های مختلفی می‌توان به یک گراف (درخت) تبدیل کرد؛ برای مثال همان برنامه را می‌توان به شکل زیر هم مدل کرد:




## برنامه‌ریز (Scheduler)

برنامه‌ریز همان برنامه‌ای است که در سیستم عامل کامپیوتر شما اجرا می‌شود تا سیستم بتواند به‌سرعت بین دو یا چند برنامه سویچ کند و حس همزمانی اجرای آنها را القا کند. در واقع برنامه‌ریز مقدار کمی از هر برنامه را اجرا می‌کند و این کار را تکرار می‌کند. انواع مختلفی از برنامه‌ریز وجود دارد که جزئیات روش کارشان متفاوت است.

گفتیم برنامه‌ریز «مقدار کم»ی از هر برنامه را اجرا می‌کند. معنی این «مقدار کم» دقیقاً چیست؟ این جا است که با مفهوم ریسمان آشنا می‌شویم. خواهیم دید که برنامه‌ریز از هر ریسمان می‌خواهد یک «قدم کوچک» از کار خود را اجرا کند و تصمیم‌گیری درباره‌ی معنی «قدم کوچک» را به‌عهده‌ی ریسمان می‌گذارد.

### ریسمان (Thread)

ریسمان‌ها مسئولیت اجرای یک برنامه را (در قدم‌هایی کوچک) برعهده دارند. در انواع مختلف ریسمان ممکن است روند اجرای برنامه اندکی متفاوت باشد.

به‌طور کلی، یک ریسمان برنامه‌ای را که باید اجرا کند در قالب فانکتور (تابع) دریافت می‌کند. سپس ما این ریسمان را - به‌همراه ریسمان‌های دیگر که هرکدام یک برنامه در خود دارند - به یک برنامه‌ریز می‌دهیم. هنگامی که برنامه‌ریز از ریسمان ما می‌خواهد که «مقدار کمی» از برنامه را اجرا کند، ریسمان یکی از گره‌های قابل‌اجرای فانکتور را اجرا می‌کند. یک گره از فانکتور هنگامی قابل اجرا است که تمامی گره‌های فرزند آن قبلاً اجرا شده باشند. اگر اجرای برنامه‌ی ریسمان کامل شود، (شرط پایان اجرا در ادامه گفته خواهدشد) آن ریسمان باید از لیست ریسمان‌های برنامه‌ریز بیرون برود.

## بخش اول - صورت مسأله (کتاب‌خانه‌ی ریسمان‌ها)

در بخش اول این پروژه قصد داریم که اجرای تعدادی ریسمان را که هر کدام مسئول محاسبه‌ی یک تابع هستند، در قالب یک کتابخانه[۱]ی ریسمان، شبیه‌سازی کنیم. به این منظور انواع برنامه‌ریزها و ریسمان‌ها را در ادامه توضیح می‌دهیم. شما باید هرکدام از این‌ها را (با طراحی درست و مبتنی بر اصول) پیاده‌سازی کنید. **در پایان این بخش باید بتوانید با استفاده از کتابخانه‌ی طراحی شده برنامه‌ی ساده‌ای را اجرا کنید. توجه کنید که برای طراحی کلاس‌ها باید از مفاهیم وراثت و چندریختی بودن که با آن‌ها آشنا شده‌اید استفاده کنید.**

### انواع برنامه‌ریزها

در این پروژه ما قصد داریم سه نوع برنامه‌ریز را پیاده‌سازی کنیم:

روش اول برنامه‌ریز تصادفی[۳] است که به‌صورت تصادفی یکی از ریسمان‌ها را برمی‌گزیند.

روش دوم پیاده‌سازی دوره‌ای[۲] است. اگر فرض کنیم یک شماره‌گذاری فرضی بر روی ریسمان‌ها (بر اساس ترتیب ورود آن‌ها به صف برنامه‌ریز) وجود دارد، ابتدا ریسمان اول انتخاب و مقدار کمی از آن اجرا می‌شود، سپس ریسمان دوم و به همین شکل تا ریسمان آخر ادامه پیدا می‌کند و پس از تمام شدن لیست اجرا مجدداً از ابتدا شروع می‌شود.

روش سوم بر اساس اولویت[۳] ریسمان‌هاست. ما می‌توانیم هنگام ایجاد ریسمان‌ها به آن‌ها اولویتی در بازه‌ی صفر تا بیست اختصاص دهیم. در برنامه‌ریزی بر اساس اولویت، هرچه اولویت یک ریسمان بیشتر باشد شانس انتخاب شدن آن بیشتر است. برای مثال، اگر سه ریسمان با اولویت‌های ۱، ۱ و ۴ در سیستم باشند، دو ریسمان اول هر کدام به احتمال $\frac{1}{6}$ و ریسمان آخر به احتمال $\frac{4}{6}$ انتخاب می‌شوند.

---

۱ Library

۲ Round-Robin

۳ Priority

## عملکرد انواع ریسمان

یک ریسمان باید بداند در هر لحظه چگونه مقدار کم را تعریف کند. برای این کار روش‌های مختلفی وجود دارد.

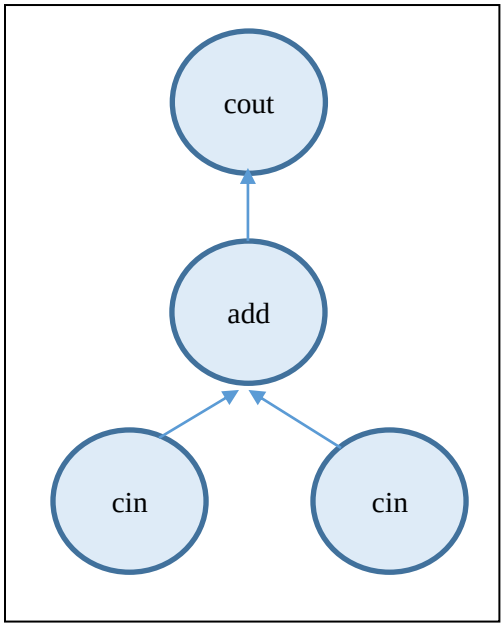
هر ریسمان باید با استفاده از الگوریتم DFS (پیمایش عمق‌اول گراف) اولین گره قابل اجرا را از گراف برنامه‌ی (functor) خود انتخاب کند و آن را به یکی از شیوه‌هایی که در ادامه گفته می‌شود اجرا کند. توجه داشته باشید که گره انتخاب‌شده باید یکی از گره‌های درخت باشد که تمام بچه‌های آن قبلاً اجرا شده‌اند؛ مثلاً نمی‌توانیم a را که برابر ۲ هست با b که حساب نشده‌است جمع کنیم و باید اول b را حساب کنیم.

همچنین یک ریسمان باید بداند که پس از انتخاب کردن و محاسبه یکی از گره‌های خود، با آن چه کند. در این کتاب‌خانه باید سه نوع رسیمان زیر را پیاده‌سازی کنید.

- در حالت کلی هر ریسمان پس از اجرای گره انتخاب‌شده، کنترل پردازنده را به برنامه‌ریزی که ریسمان را اجرا کرده بود باز می‌گرداند تا برنامه‌ریز بتواند ریسمان دیگری را انتخاب کند؛ ولی این روند گاهی پیچیده‌تر است.
- یکی از انواع ریسمان که باید پیاده‌سازی کنید ریسمانی است که اجرای آن تا پایان محاسبه و اجرای کل درخت برنامه متوقف نمی‌شود، یعنی هر زمان که اجرای آن شروع شود به جای این که تنها یک گره حساب شود، کل درختِ ریسمان محاسبه می‌شود.
- نوع دیگر حالت بازگشتی است؛ یعنی هر زمان که محاسبه‌ی درخت تمام شد مجدداً همان درخت را محاسبه کند. دقت کنید که اگر محاسبه‌ی ریسمانی کامل شود برنامه‌ریز آن را از لیستش خارج می‌کند، ولی در این حالت چنین اتفاقی نباید رخ دهد.

## تست و اجرا

برای اطمینان از درستی کتابخانه‌ی خود **دو** برنامه بنویسید که دو عدد را از ورودی بخوانند و یک برنامه مجموع آن‌ها و برنامه‌ی دیگر ضرب آن‌ها را در خروجی استاندارد بنویسد. گراف برنامه باید به شکل زیر باشد. در مرحله‌ی اول صرفا سعی کنید برنامه را بدون برنامه‌ریز اجرا کنید و در مرحله‌ی بعد دو برنامه را به برنامه‌ریزها بدهید و سعی کنید به کمک آن‌ها برنامه را اجرا کنید.



## بخش دوم استفاده از کتاب‌خانه‌ی ریسمان‌ها

پس از پیاده‌سازی کتابخانه‌ی گفته شده، شما باید برنامه‌ای بنویسید که از آن استفاده کند. به همین منظور برنامه‌ی شما باید بتواند:

- درخت‌های محاسبات را به‌عنوان ورودی از فایل بخواند.
- ریسمان متناظر با درخت خوانده شده را تشکیل دهد.
- ریسمان‌های تشکیل‌شده را به برنامه‌ریز بدهد، آن‌ها را اجرا و خروجی را چاپ کند.

### تشکیل درخت محاسبات

به‌این‌منظور فرض می‌کنیم تمامی اعمالی که در درخت انجام می‌شوند در یکی از انواع زیر قرار می‌گیرند و برنامه‌هایی (functor) که در فایل ورودی مشخص خواهند شد به کمک ترکیب این گره‌های اولیه تشکیل می‌شوند.

- بدون‌ورودی: (۱) خواندن از ورودی استاندارد، (۲) نوشتن در خروجی استاندارد
- تک‌ورودی: (۳) عملیات قرینه‌کردن
- دوورودی: (۴) جمع و (۵) ضرب
- سه‌ورودی: (۶) انتخاب یکی از ورودی‌های دوم(b) یا سوم(c)، به‌عنوان خروجی، براساس ورودی اول (a). (if(a) return b; else return c;)

### اجرای ریسمان‌ها

برنامه باید ریسمان‌هایی را که در مرحله‌ی قبل خوانده به یک برنامه‌ریز بدهد و از برنامه‌ریز بخواهد آن‌ها را اجرا کند. نوع برنامه‌ریز به عنوان پارامتر کامندلاین در هنگام اجرا به برنامه‌ی شما داده می‌شود. که یکی از مقادیر زیر خواهد بود.

- random
- roundrobin
- priority

### خواندن فایل ورودی با فرمت JSON

JSON (مخفف JavaScript Object Notation) یک استاندارد متنیِ سبک برای انتقال داده‌ها است، به‌گونه‌ای که برای انسان نیز خوانا باشد. در این تمرین از نسخه‌ی ساده‌شده‌ی JSON استفاده خواهیم کرد. این فرمت به‌صورت بازگشتی پیاده‌سازی می‌شود و برای نمایش اطلاعات در آن، از عناصر زیر استفاده می‌شود:

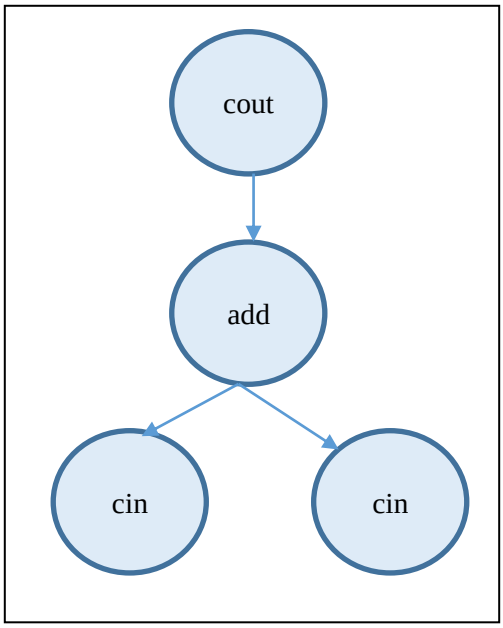
- Number: 1
- String: text
- Array: [1, 2, 3]

- Object لیست بدون ترتیبی از جفت‌های کلید و مقدار
  {key_1 : 1 , key_2 : 2 , key_3 : 3}
  - به عبارتی می‌توان فرض کرد که این المان خود یک JSON کامل است.

اطلاعات درخت‌ها به صورت زیر داده می‌شود:

در این مثال یک ترد داریم که از نوع basic_thread است و فانکتور آن به شکل زیر است و پس از خواندن دو ورودی از ورودی استاندارد خروجی را چاپ می‌کند.

```
{
  threads : [
    { thread_name : addition ,
      thread_type : basic_thread ,
      functor : {
       functor_type : cout_num ,
       functor_child : [
        {
          functor_type : add ,
          functor_child : [
            { functor_type: cin_num },
            { functor_type: cin_num }
          ]
        }
      ]
    }
    }
  ]
}
```




نکات:

- threads یک آرایه از ریسمان‌ها است که هر ترد به شکل یک Object در این آرایه است.
- نام ترد هر چیزی می‌تواند باشد و صرفا می‌تواند برای بررسی و دیباگ برنامه کاربرد داشته باشد. (thread_name)
  - برای تست برنامه زمانی که اتفاقی در سطح ریسمان می‌افتد آن را همراه با نام ریسمان گزارش کنید. مثلا این که اجرای ریسمان شروع شد و یا پایان یافت.
- thread_type نوع ترد را مشخص می‌کند که می‌تواند یکی از سه نوع زیر باشد:
  - basic_thread
  - recurrent_thread
  - singlerun_thread
- قسمت functor مشخص کننده‌ی برنامه‌ی ریسمان است که مقدار آن از نوع object است و نشان‌دهنده‌ی یک گره (گره ریشه‌ی درخت محاسبه) است.
- functor_type مشخص کننده‌ی نوع گره محاسباتی است که در قسمت «تشکیل درخت محاسبات» تعریف شدند. این قسمت می‌تواند یکی از مقادیر زیر را داشته باشد.
  - cout

- cin
- add
- mul
- neg
- if

- functor_child یک آرایه است که مجموعه‌ی گره‌هایی که فرزند گره فعلی در گراف هستند را مشخص می‌کند. هر کدام از اعضای این آرایه خود یک Object هستند و در واقع به صورت بازگشتی درخت محاسبات مشخص می‌شود. همچنین دقت کنید که اگر گره‌ای فرزندی نداشته باشد، مشخصه‌ی functor_child آن وجود نخواهد داشت.

### تست و اجرا

با فرض این که نام فایل اجرایی برنامه‌ی شما thread.out است، برنامه به شکل زیر اجرا خواهد شد:

```
./thread.out {JSON File} {Scheduler Type}
```

برای مثال:

```
./thread.out program.json roundrobin
```

## بخش امتیازی

**توجه: تنها در صورتی که همه‌ی پروژه را به صورت کامل پیاده سازی کرده‌اید این قسمت را مطالعه کنید.**

در قسمت انواع ریسمان‌ها توضیح دادیم که هر ریسمان ابتدا گره‌ای را از درخت محاسبات انتخاب می‌کند و سپس با توجه به نوع خود آن را اجرا می‌کند. در این قسمت می‌خواهیم روش انتخاب گره دیگری را به پروژه اضافه کنیم.

در قسمت قبل تنها راه انتخاب گره از گراف برنامه به وسیله‌ی جستجوی DFS بود. اکنون می‌خواهیم که یک روش دیگر نیز اضافه کنیم. این روش انتخاب تصادفی می‌باشد (که البته همچنان باید به وابستگی گره‌ها توجه کرد). یعنی هرگاه برنامه‌ریز از ریسمان میخواهد مقدار کمی از خود را اجرا کند، ریسمان به صورت تصادفی یکی از گره هایی که **قابل اجرا** هستند را انتخاب می‌کند.

چه راهی برای پیاده‌سازی این تغییر به ذهن شما می‌رسد؟ ابتدا سعی کنید خودتان طراحی خوبی برای این مساله پیدا کنید. در صورتی که علاقه‌مندید پیشنهاد می‌شود مطالبی را در مورد الگوی طراحی استراتژی[4] مطالعه کنید. الگوهای طراحی از مطالب پیشرفته‌تری در زمینه‌ی طراحی است که در سال‌های آینده و به مرور با آن‌ها آشنا خواهید شد.

4 https://en.wikipedia.org/wiki/Strategy_pattern

## نکات پایانی

- طراحی شما دراین تمرین از اهمیت ویژه‌ای برخوردار است. از مفاهیم وراثت و چندریختی‌بودن استفاده کنید.
- در هنگام پیاده سازی پروژه ابتدا بخش اول را به صورت کامل انجام دهید و سپس سراغ قسمت دوم بروید.
- پروژه‌ی شما باید multifile باشد و حتما باید از Makefile استفاده کنید.
- در صورت وجود ابهام، ابتدا متن پروژه را دقیق مطالعه کنید و اگر ابهام برطرف نشد در فروم درس سوالات خود را مطرح نمایید.
-

## نحوه‌ی تحویل

فایل‌های مربوط به برنامه‌ی خود را در پوشه‌ای با نام A6-SID.zip در سایت درس آپلود کنید. (SID پنج رقم آخر شماره‌ی دانشجویی شماست. به عنوان مثال اگر شماره‌ی دانشجویی شما ۸۱۰۱۹۴۱۲۳ است، نام فایل شما باید A۶-۹۴۱۲۳.zip باشد.) فایل آپلودی شما باید شامل پوشه‌ی کامل پروژه باشد.

تحویل این تمرین به صورت حضوری است و در هنگام تحویل باید به تمام قسمت‌های کد خود مسلط باشید.

## دقت کنید

- برنامه‌ی شما باید در سیستم‌عامل لینوکس نوشته و با کامپایلر g++ کامپایل شود.
- به فرمت و نام فایل های خود دقت کنید. در صورتی که هر یک از موارد گفته شده رعایت نشود، نمره‌ی صفر برای شما در نظر گرفته می‌شود.
- در صورت کشف تقلب در کل و یا قسمتی از تمرین، برای هر دو طرف نمره‌ی ۱۰۰- منظور خواهد شد.