دانشگاه تهران

پروژه پایانی مبانی برنامه‌نویسی دانشکده آمار، ریاضی و علوم کامپیوتر زمان تحویل: ۲۶ بهمن

زبان ساده‌ای برای نقاشی به نام SimplePaint معرفی می‌کنیم.
گزاره‌های این زبان متشکل از جمع، تفریق، تقسیم، باقی‌مانده و ضرب بین اعداد مثبت و صفر یا توابعی که تا محل ظهور گزاره به صورت کامل تعریف شده‌اند هستند.
در خط اول این زبان دو عدد که با فاصله جدا شده‌اند به ترتیب نماینده ارتفاع و عرض تصویر هستند و باقی خطوط میتوانند خالی یا شامل توصیف تابع باشند.
در هر برنامه این زبان تابعی با نام main که هیچ ورودی‌ای نمی‌گیرد وجود دارد و برنامه‌هایی که این تابع را ندارند یا تابع main آنها ورودی دارد برنامه‌ای معتبر در این زبان نیستند.

توابع پایه‌ای زبان:

```
if(cond, t, f) = t if cond != 0 else f
drawPoint(x, y, r, g, b) = 0
drawLine(x0, y0, x1, y1, r, g, b) = 0
drawCircle(x, y, radius, r, g, b) = 0
```

سه آرگومان r و g و b در سه تابع $draw$ نماینده رنگ هستند و باید در بازه ۰ تا ۲۵۵ باشند و صدا شدن آنها با مقادیری دیگر باعث خطا در زمان اجرا می‌شود.

تعریف تابع:

```
func functionName([arg1[,arg2[,arg3...]]]) expr
```

تعریف تابع بازگشتی:

```
rfunc functionName([arg1[,arg2[,arg3...]]], rArg)
    0 baseExpr
    rVal rExpr
```

توصیف تابع بازگشتی از دو بخش تشکیل شده‌است، گزاره اول که بعد از عدد ۰ و یک فاصله می‌آید که دسترسی به آرگومان‌های تابع به جز $rArg$ دارد نمایش‌دهنده مقدار تابع برای مقدار $rArg$ صفر است و گزاره دوم که بعد از $rVal$ می‌آید با دسترسی به آرگومان‌های تابع شامل $rArg$ و همینطور $rVal$ که نماینده مقدار تابع با همان آرگومان‌ها برای مقدار $rArg$ است نمایش‌دهنده مقدار تابع با همان آرگومان‌ها و $rArg + 1$ است.
به بیانی دیگر

```
functionName([arg1[,arg2[,arg3...]]], 0) = baseExpr([arg1[,arg2[,arg3
    ...]]])
functionName([arg1[,arg2[,arg3...]]], rArg + 1) = rExpr(rVal=functionName([
    arg1[,arg2[,arg3...]]], rArg), rArg=rArg, [arg1[,arg2[,arg3...]]])
```

نام‌های $rArg$، $rVal$، $functionName$، $arg1$، $arg2$ و ... صرفا نمادین هستند و در زبان هرچیزی می‌توانند باشند.
همینطور در تعریف توابع دقت کنید که نام هیچ آرگومان و همینطور هیچ تابعی نمیتواند draw، drawPoint، if، func، drawLine، drawCircle و rfunc باشد. همینطور نام توابع و آرگومان ها متشکل از حروف انگلیسی کوچک و بزرگ و عدد است و حتما با حرف انگلیسی شروع می‌شود. دقت کنید که دو گذاره بعد از خط شامل rfunc حتما باید ۴ فاصله جلوتر باشند و بین آنها و همینطور خط شامل rfunc خط خالی‌ای وجود نداشته باشد.
ورودی منفی در آرگومان آخر یک تابع بازگشتی نامعتبر است و باعث خطا در زمان اجرا می‌شود.

تجزیه یک برنامه SimplePaint در قالب JSON :
تجزیه اعداد در JSON برابر با همان عدد است.
جمع:

```
expr1 + expr2
```

```
{
  "type": "+",
  "A": expr1P,
  "B": expr2P
}
```

که در تجزیه JSON مقادیر $expr1P$ و $expr2P$ به ترتیب برابر با تجزیه دو گزاره $expr1$ و $expr2$ هستند.

تجزیه عملگرهای تفریق، تقسیم، ضرب و باقی‌مانده نیز به صورت مشابه و به ترتیب با $type$های "$-$"، "/"، "$*$" و "%" هستند.
تجزیه آرگومان‌ها برابر با رشته‌ای مساوی با نام آن‌ها است.

صدا شدن یک تابع:

```
functionName(arg1, arg2, arg3, ...)
```

```
{
  "type": "function call",
  "function name": "functionName",
  "args": [
    arg1P,
    arg2P,
    arg3P,
    ...
  ]
}
```

تعریف تابع:

```
func functionName([arg1[,arg2[,arg3...]]]) expr
```

```
{
  "type": "function definition",
  "function name": "functionName",
  "args": [
    "arg1",
    "arg2",
    "arg3",
    ...
  ],
  "expression": exprP
}
```

تعریف تابع بازگشتی:

```
rfunc functionName([arg1[,arg2[,arg3...]]], rArg)
    0 baseExpr
    rVal rExpr
```

```
{
  "type": "recursive function definition",
  "function name": "functionName",
  "args": [
    "arg1",
    "arg2",
    "arg3",
    ...
  ],
  "recursive arg": "rArg",
  "base expression": baseExprP,
  "recursive expression": {
    "recursive value name": "rVal",
    "expression": rExprP
  }
}
```

برنامه:

```
height width

func func1 ...

func func2 ...

rfunc func3 ...
```

```
{
  "height": 100,
  "width": 200,
  "functions": [
    func1P,
    func2P,
    func3P
  ]
}
```

پروژه متشکل از دو برنامه کلاینت و سرور است.
برنامه کلاینت در آرگومان ورودی خود نام فایل برنامه‌ای به زبان SimplePaint را می‌گیرد، در صورتی که محتویات فایل یک برنامه معتبر نباشد اعلام می‌کند. در صورت معتبر بودن برنامه آن را تجزیه به JSON می‌کند و به از طریق یک درخواست HTTP برای سرور ارسال می‌کند. سرور در جواب یک شناسه کار به کلاینت می‌دهد، کلاینت باید منتظر بماند تا کار با آن شناسه پایان پذیرد و نتیجه را که یک تصویر است از سرور دانلود کند.

برنامه سرور از دو بخش تشکیل شده‌است. بخش اول یک سرور HTTP که جواب کلاینت را بدهد و بخش دوم پروسه‌های کارگر که برنامه‌هایی که از سمت کلاینت می‌آید را اجرا کند و تصویر را بسازد تا کلاینت بتواند دانلود کند.

سرور شامل endpoint های ریز است:

1.
```
POST /job/
```

2.
```
GET /job/<job-id>/
```

3.
```
GET /media/<filename>
```

endpoint اول زمانی صدا خواهدشد که کلاینت برنامه جدیدی برای اجرا به سرور فرستاده است و JSON برنامه در بدنه درخواست HTTP خواهدبود. در جواب شناسه کار ساخته‌شده متناظر با برنامه باید برگردانده شود.
endpoint دوم وضعیت کار با شناسه $job-id$ را برمی‌گرداند، این وضعیت در صورتی که کار تمام شده باشد لینک دانلود و در غیر این صورت پیغامی متناسب است. در صورت بروز خطا در زمان اجرا این endpoint باید در جواب این موضوع را اعلام کند. endpoint سوم وضیفه شروع دانلود برای کلاینت برای فایل درخواستی را دارد.

دقت کنید پروسه‌های کارگر سرور باید جدا از سرور وب باشند و تصویر در مسیر جواب درخواست HTTP ساخته نشود.

چند نکته:

- عملگر تقسیم معادل کف تقسیم است و مقدار صحیح کف حاصل از تقسیم را برمیگرداند.
- دقت کنید که فقط در مورد تابع if نباید هر دو گزاره true و false را محاسبه کنید و فقط گزاره‌ای که قرار بر برگردانده شدن است را اجرا کنید، وگرنه عملکرد if از بین خواهد رفت.

امتیازی:

- پیاده‌سازی تابع پایه‌ای اضافه $drawEllipse(x0, y0, x1, y1, r, g, b)$
- بازگرداندن درصد جلو رفتن کار در endpoint دوم سرور و نمایش آن در کلاینت.
- ارسال محل رخداد خطا اجرا.

موفق باشید.