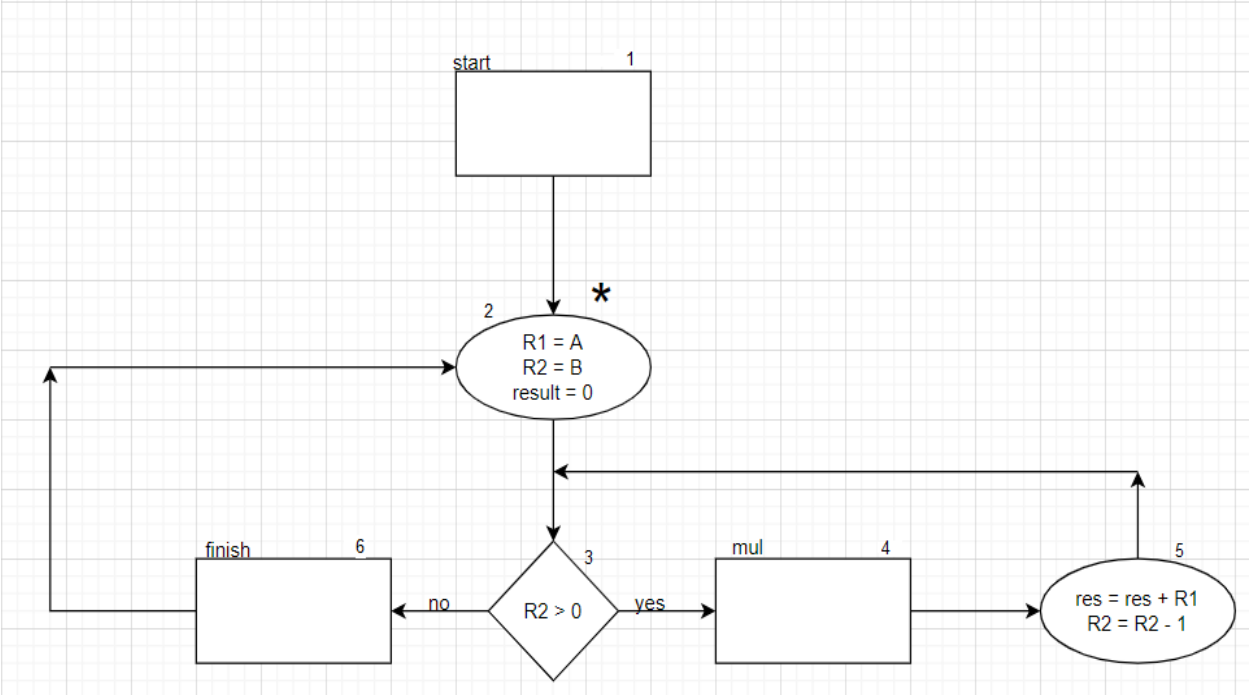
ورودی این برنامه به شکل json است که به شکل دستی این json باید ساخته شود و script بر روی آن ران شود.

قواعد json:

اول از همه یک object کلی باید گذاشته شود. به شکل {}.

سپس به ازای هر شکلی که داخل asm chart می خواهیم داشته باشیم، باید یک object دیگر داخل این object اصلی داشته باشیم. نام این object ها که به ازای هر نود وارد object اصلی می شود هم برای مستطیل باید با rectangle، برای لوزی باید با diamond و برای بیضی با obal شروع شود و چون کلید های یکسان نباید داشته باشیم، بعد از این کلید واژه ها می توان مثلا یک عدد به عنوان نشان گذاشت مثلا rectangle1 و oval2 و ... که در مثال زیر این مورد نشان داده شده است. حال قواعد ساخت هر کدام از شکل ها را بیان می کنم. سعی می شود روی مثال توضیح داده شود تا بهتر متوجه شویم:



قبل از شروع، دقت شود که یک و فقط یک شکل را باید با علامت ستاره مشخص کرد تا این شکل به عنوان stop point ها باشد. خب در asm chart ها همواره بعد از اجرای یکباره ی الگوریتم، به حالت ابتدایی بر می گردیم و اگر در این بخش stop point نداشته باشیم، در loop بی نهایت مدام asm chart اجرا می شود. پس داخل object مربوط به یک شکل با عبارت

“star”:true

می توان گفت که این شکل همان شکل star دار ما باشد.

در ضمن هر شکل یک id هم باید داشته باشد تا با کمک این id بتوان ارتباط شکل ها به همدیگر را مشخص کرد که این مورد هم در شکل مثال مشخص است.

مستطیل:

همانطور که می دانیم، این حالت نشان دهنده یک state است. این state یک id دارد و یک name دارد و با کلید next هم می توان مشخص کرد که نود بعدی که بعد از این نود به آن نود باید برویم، کدام است. مورد دیگری برای این شکل مورد نیاز نیست. تا اینجای json مانند عکس زیر است:

در next هم باید id نود بعدی را بگذاریم.

بیضی:

برای این حالت باز هم یک id نیاز است و یک کلید next هم نیاز است تا مشخص کند بعد از این نود به کدام نود باید برویم. اما می دانیم که درون بیضی است که عملیات های اصلی مشخص می شود. برای نوشتن عملیات، دقت کنید که باید یک کلید statements مشخص کنید و value این کلید را از نوع لیست قرار دهید [ ] و سپس به شکل string عبرت ها را داخل این string بنویسید. دقت کنید که داخل هر عبارت، یک عملیات باید انجام شود یعنی مثلا اگر a + b * c را می خواهیم انجام دهیم، نمی توانیم بنویسیم

[“x = a + b * c”]

بلکه باید در ۲ عبارت جداگانه آن را بنویسیم یعنی به شکل زیر:

[“y = b * c”, “x = a + y”]

نکته بعدی این است که اسم متغیر ها نباید space داشته باشد و بین هر علامت ریاضی و هر متغیر باید یک space گذاشته شده باشد تا برنامه به درستی بتواند عملیات split کردن را انجام دهد.

نکته بعدی این است که سمت چپ تساوی باید یک متغیر باشد ولی سمت راست تساوی می تواند متغیر و یا عدد باشد. از علامت های ریاضی هم استفاده کنید و از عبارت های خلاصه شده ی =- و یا =* که در زبان های برنامه نویسی موجود است، استفاده نفرمایید.

برای عملیات های ریاضی، می توانید از =، +، -، *، /، و ٪ استفاده کنید و برای مقایسه ها م=هم از هر کدام از حالت های ==، =!، >، <، =>، =< می توانید استفاده کنید.

در ضمن درون بیضی از operator های باینری >>، <<، &، | و ^ هم می توان استفاده کرد.

دقت شود که value مربوط به کلید statements حتما باید list باشد. حتی اگر تنها یک statements داخل آن انجام می شود. اگر هیچ کاری هم داخل آن قرار نبود انجام شود، یک لیست خالی قرار دهید.

بعد از وارد کردن بیضی با آیدی در مثال، فایل json مانند عکس زیر می شود:

لوزی:

این شکل هم یک id دارد و شرط مد نظر باید در کلید statements نوشته شود. این شرط هم باید مثل حالت بیضی، با یک space بین متغیر و یا عدد با علامت مدنظر بیاید. در این حالت دو طرف علامت می تواند متغیر و یا عدد باشد و تفاوتی ندارد و همانطور که گفته شد از ۶ حالت ==، =!، >، <، =>، =< می توان استفاده کرد. در این حالت هم با next باید نود بعدی را مشخص کرد ولی تفاوتی که دارد این است که یک next_positive داریم و یک next_negative که مشخص می کند در صورت true یا false شدن عبارت، به کجا باید برویم.

در این مورد دقت کنید که value مربوط به کلید statements نباید لیست باشد و یک تک دستور به شکل string باید باشد. یعنی دستور های تو در تو، با چند لوزی پشت هم باید اجرا شود.

در مثال گفته شده، به این شکل وارد json می کنیم:

```json
{
  "rectangle1": {
    "id": 1,
    "name": "start",
    "next": 2
  },
  "oval1": {
    "id": 2,
    "next": 3,
    "star": true,
    "statements": [
      "R1 = 10",
      "R2 = 4",
      "result = 0"
    ]
  },
  "Diamond1": {
    "id": 3,
    "next_positive": 4,
    "next_negative": 6,
    "statements": "R2 > 0"
  }
}
```

به همین ترتیب برای هر نود دیگر هم اطلاعات را وارد می کنیم و در نهایت فایل json به شکل زیر می شود.

```
{
  "rectangle1": {
    "id": 1,
    "name": "start",
    "next": 2
  },
  "oval1": {
    "id": 2,
    "next": 3,
    "star": true,
    "statements": [
      "R1 = 10",
      "R2 = 4",
      "result = 0"
    ]
  },
  "Diamond1": {
    "id": 3,
    "next_positive": 4,
    "next_negative": 6,
    "statements": "R2 > 0"
  },
  "rectangle2": {
    "id": 4,
    "name": "mul",
    "next": 5
  },
  "oval2": {
    "id": 5,
    "next": 3,
    "statements": [
      "result = result + R1",
      "R2 = R2 - 1"
    ]
  },
  "rectangle3": {
    "id": 6,
    "name": "finish",
    "next": 2
  }
}
```

به این ترتیب json مدنظر ساخته می شود و حال کافی است که آدرس آن را داخل script درست کنید و کد تحلیل asm chart را برای شما انجام می دهد.
در نهایت دقت کنید که یک node به عنوان نود آغازین باید در نظر گرفته شود که با
"start”:true
می توان این مورد را مشخص کرد و در مثال ما، همان rectangle1 نود آغازین است به آن، این عبارت را اضافه می کنیم. طبیعتا باید از یک state کار asm chart را شروع کنیم پس بهتر است که برای عملکرد و تحلیل بهتر، این کلید را در یک نود از نوع rectangle بگذاریم.