Context

Panoply dynamically inserts advertisements into podcasts when they are delivered to listeners. When a podcaster uploads an episode, they add insertion points to mark where ads should be placed within the episode.

The Problem

You will be writing some code in the language of your choosing to determine which ads to insert into a given episode audio file. Given an episode and data about the ad campaigns that are currently running, you will give back the audio for the episode with the ads inserted. Any slots that are not filled with ads will simply be stripped from the audio file representation.

Structure of data

This exercise does not involve any real audio files. Instead, audio files will be represented by strings in the format:

`[PRE]++++++++++[MID]++++[MID]++++++++++[POST]`

where each `+` is an audio frame in the file, and `[PRE]`, `[MID]`, or `[POST]` signify an insertion point for that type of advertisement. For context there are 3 types of ads we place: pre rolls (at beginning of audio file), mid rolls (in the middle of the audio file), and post rolls (at end of the audio file). There may be none or several of each of these in a given episode audio file.

Audio for ads will also be represented by strings like: `*Acme*`.

The **episode** will be represented in a key value structure (or a similiar structure in your implementation language):

```
{ audio: "[PRE]+++++[MID]+++++", id: "dag-532" }
```

where id is a unique identifier used by ads to target specific episodes

Each **ad** will be represented in the key value structure:

```
{ audio: "*Acme*", type: "PRE", targets: ["afs-354", "dag-523"], revenue: 10 }
```

where the targets are the episode ids that it may be placed in and the revenue is the revenue earned per listen

Ads are part of **ad campaigns**, which is just a list of one or more ads that must be placed together. So if there is an ad campaign with 2 mid rolls and the episode only has room for 1 midroll, it will not be placed in the episode.

Ad Insertion Rules

Try to fill each ad spot in the episode until there are no insertion points remaining (or no remaining campaigns that can fill the remaining slots). Any remaining insertion points that cannot be filled may be stripped.

Ad slots must be filled by the correct type of ad (eg `PRE`) and an ad may only be used in an episode if the episode is in its targets (you may assume that each ad has identical targets within an ad campaign).

Each ad in a particular ad campaign must be used in the episode or none from that campaign at all. For this reason, there may be several possible correct solutions depending on what ad campaigns are inserted first.

Bonus: Optimize such that the revenue provided by the ads is maximized.

Test Data

While you may structure your code however you'd like, we will provide you with some example audio and ad data below:

Example Episodes

```
{ audio: "[PRE]+++++[MID]+++++", id: "dag-532" }
```

```
{ audio: "+++++[MID]++++[MID]++++[POST]", id: "kld-412" }
```

```
{ audio: "[PRE][PRE]+++++[MID]+++[MID]+++++[POST]", id: "abc-444" }
```

Example Ad Campaigns

```
[
  { audio: "*AcmeA*", type: "PRE", targets: ["dag-523"], revenue: 12 },
  { audio: "*AcmeB*", type: "MID", targets: ["dag-523"], revenue: 4 },
  { audio: "*AcmeC*", type: "MID", targets: ["dag-523"], revenue: 14 }
]
```

```
[
  { audio: "*CorpCorp*", type: "POST", targets: ["afs-354", "dag-523"], revenue: 3 }
]
```

Example Inputs and Outputs

Given the currently running ad campaigns:

```
[
  { audio: "*AcmeA*", type: "PRE", targets: ["dag-892", "hab-812"], revenue: 1 },
  { audio: "*AcmeB*", type: "MID", targets: ["dag-892", "hab-812"], revenue: 4 },
  { audio: "*AcmeC*", type: "MID", targets: ["dag-892", "hab-812"], revenue: 5 }
]
```

```
[
  { audio: "*TacoCat*", type: "MID", targets: ["abc-123", "dag-892"], revenue: 3 }
]
```

```
[
  { audio: "*CorpCorpA*", type: "PRE", targets: ["abc-123", "dag-892"], revenue: 11 },
  { audio: "*CorpCorpB*", type: "POST", targets: ["abc-123", "dag-892"], revenue: 7 },

]
```

```
[
  { audio: "*FurryDogA*", type: "PRE", targets: ["dag-892", "hab-812", "efa-931"], revenue: 11 },
  { audio: "*FurryDogB*", type: "PRE", targets: ["dag-892", "hab-812", "efa-931"], revenue: 7 },
```

```
]
```

```
[
  { audio: "*GiantGiraffeA*", type: "MID", targets: ["paj-103", "abc-123"], revenue: 9 },
  { audio: "*GiantGiraffeB*", type: "MID", targets: ["paj-103", "abc-123"], revenue: 4 },

]
```

Episode	Possible Audio Output
`{ id: "dag-892", audio: "++++[MID]++++[MID]++++[POST]" }`	`+++++TacoCat*********`
`{ id: "abc-123", audio: "[PRE]++++[MID]++++[MID]++[MID]++[POST]" }`	`*CorpCorpA*+++++TacoCat*+++++GiantGiraffeA*+++GiantGiraffeB*+++CorpCorpB*`
`{ id: "hab-812", audio: "[PRE][PRE]++++[MID]++++[MID]++[MID]++[POST]" }`	`*AcmeA*+++++AcmeB*+++++AcmeC*+++++`
`{ id: "efa-931", audio: "[PRE][PRE]++++++++++" }`	`*FurryDogA**FurryDogB*++++++++++`
`{ id: "paj-103", audio: "++++[MID]+++++[MID]++++[MID]++[POST]" }`	`+++++GiantGiraffeA*++++++GiantGiraffeB*++++++`

Requirements

Given an episode and a number of ad campaigns that are ongoing, fill the episode's audio with ads from the ad campaigns following the logic described above, and return the audio file, now filled with ads.

You may structure your program however you wish and provide an API that makes sense in the context of your solution though it may be as simple as a function, (episode, adCampaigns) -> episodeAudioWithAds. **Your solution must run in the terminal or browser console.** At Panoply, you'd be working on the full-stack, but for this challenge a web app is not necessary.

You may complete the exercise in whichever language you feel most comfortable. Try to time box yourself to 90 minutes, though it may not even take you that much time to complete. If you use any code snippets that are not your own please notate them with a comment.

Send us back relevant code as well as any instructions or description about your approach that you deem relevant or helpful for us reviewing your solution. Please attach a Git repo with your code via e-mail to your Panoply contact.