STORY 1 定义并解析 FLAGS

在命令行应用中我们经常遇到 flag/switch 参数。这些参数无需设置后续的值，往往会起到 boolean 变量的作用。例如：

```
rm -r ...
```

其中的 -r 就是 flag。表明这次删除操作是递归操作。 由于 flag 是 boolean 变量，因此在 **我们的库** 中是不允许重复设定的。例如

```
ruby --version --version
```

是一个不合法的参数列表。

这个 Story 期望我们的命令行应用程序解析库可以解析类似这种含有单一的 flag 参数的参数列表。其使用方式大概是这样的：

```
ArgsParser parser = new ArgsParserBuilder()
    .AddFlagOption("flag", 'f', "This is a description")
    .Build();

ArgsParsingResult result = parser.Parse(new [] { "--flag" });

Assert.True(result.IsSuccess);
Assert.True(result.GetFlagValue("-f"));
Assert.Null(result.Error);
```

AC1 每一个 Flag 变量包含一个完整形式（Full Form）和简单形式(Abbreviation Form)。其中完整形式是一个字符串，而简单形式仅有一个字符。

- 每一个 Flag 至少保证有 *完整形式* 或 *简单形式* 的定义。
- 完整形式至少应该包含一个字符，并且只能够包含大小写英文字母，数字，下划线和短划线，且第一个字幕不能够为短划线。
- 简单形式只能够包含大小写英文字母。
- 如果要使用完整形式，则需要在完整形式之前加入双短划线。例如，完整形式为 `no-edit` 则在命令行应该使用 `--no-edit` 进行标识。
- 如果要使用简单形式，则需要在简单形式之前假如一个短划线。例如，简单形式为 `o` 则在命令行应该使用 `-o` 进行标识。

- 目前不支持使用连续简单形式的 flag 变量。例如 `-rf`。

AC2 每一个 Flag 还需要包含一个描述信息（Description），这个描述信息以后将用于自动进行输出提示。

- 这个描述性信息是可选的，因此可以不设置。
- 为了今后输出的方便，所有的换行符将使用空格代替。

AC3 目前一个 Parser 仅支持一个 Flag 变量的定义。不能定义多个。

AC4 Flag 是不存在是否 mandatory 这个设定的。如果出现在了命令行参数中，则为 true，如果没有出现在命令行参数中，则为 false。

AC5 如果命令行参数中包含除 Flag 定义之外的其他形式的参数，都应当视为解析失败。且解析的结果应当包含如下的信息：

- 错误的原因
- 以及造成错误的参数值。

```
ArgsParser parser = new ArgsParserBuilder()
    .AddFlagOption("flag", 'f', "This is a description")
    .Build();

ArgsParsingResult result = parser.Parse(new [] { "--flag", "-v" });

Assert.False(result.IsSuccess);
Assert.Equals(ParsingErrorCode.UndefinedOption, result.Error.Code);
Assert.Equals("-v", result.Error.Trigger);
```