简化 argparse

题目描述

刷新

Python 语言中有一个十分好用的命令行参数解析库： `argparse`

通过如下代码，用户可以为程序 `parser.py` 添加想要的命令行选项：

```
import argparse

parser = argparse.ArgumentParser()

parser.add_argument("--first", default=1, type=int, help="first number")
parser.add_argument("--second", default=1, type=int, help="second number")

args = parser.parse_args()

print(args.first)
print(args.second)
print(args.first + args.second)
```

使用时，可以通过命令行选项的名称指定相应的参数:

```
python parser.py --first 2 --second 3
```

输出：

```
2
3
5
```

本题的目标是使用 C++ 实现一个简化版的 `argparse`。核心测试代码在 `main.cpp` 文件的 `main` 函数中。测试代码的样例如下（此样例代码可以从这里 (/staticdata/1934.9ZoTmKIvbe6oiNWr.pub/Hq8NP45aCiArVXks.main.cpp/main.cpp)下载。**注意：评测使用的 `main.cpp` 和这里提供的文件在 `std::cout` 这一行略有不同，其他部分完全一致。**）

```
#include <iostream>
#include "Parser.h"

int main(int argc, char *argv[]) {
    Parser parser = Parser();

    parser.add_argument("--first", 1, "First number");
    parser.add_argument("--second", 2, "Next number");
    parser.add_argument("--third", 3, "Third number");

    char *args = new char[100];
    std::cin.getline(args, 100);

    parser.init(args);

    int a = parser.get_argument("first");
    int b = parser.get_argument("second");
    int c = parser.get_argument("third");

    std::cout << a + b * c << std::endl;

    return 0;
}
```

我们会先构造一个 `Parser` 对象，向其中添加几个命令行选项。命令行选项会以一行标准输入的形式给出，其中不包含 `./main`。然后调用 `parser.init`，其中会将这一整行输入区分成不同的命令行选项，并根据选项名称提取出参数并保存下来。当然，这里还需要进行参数不合法时的错误处理。最后我们会访问设置的命令行选项。

限制与约定

- 只考虑输入参数为**正整数**的情况，且均在 `int` 范围内。
- 你可以假设测试时的选项名称全部以 "--" 开头，并且 `get_argument` 中的参数全部是添加过的。
- 本题测例中命令行选项数量不会超过10。
- 输入字符串总长度不超过100。

评测样例

以下的样例包含了我们考虑的**所有**正常和错误情况。

- `--help` 显示帮助信息：

```
--help
```

输出：

```
usage: ./main [--help] [--first FIRST] [--second SECOND] [--third THIRD]

optional arguments:
--help   show this help message and exit
--first FIRST   First number
--second SECOND   Next number
--third THIRD   Third number
```

- 正常输出结果：

```
--first 3 --second 3 --third 2
```

输出：

```
9
```

- 未指定参数，使用默认值

```
--first 3 --third 2
```

输出：

```
7
```

- 使用未添加的参数

```
--fourth 3
```

输出：

```
usage: ./main [--help] [--first FIRST] [--second SECOND] [--third THIRD]
./main: error: unrecognized arguments: --fourth
```

- 参数未提供值

```
--first
```

输出

```
usage: ./main [--help] [--first FIRST] [--second SECOND] [--third THIRD]
./main: error: argument --first: expected an argument
```

- 参数后面不是整数 #1

```
--first --second 3
```

输出：

```
usage: ./main [--help] [--first FIRST] [--second SECOND] [--third THIRD]
./main: error: argument --first: invalid int value: '--second'
```

- 参数后面不是整数 #2

```
--first aa --second 3
```

输出：

```
usage: ./main [--help] [--first FIRST] [--second SECOND] [--third THIRD]
./main: error: argument --first: invalid int value: 'aa'
```

- 多个错误，按第一个错误输出

```
--first aa --fourth 4
```

输出：

```
usage: ./main [--help] [--first FIRST] [--second SECOND] [--third THIRD]
./main: error: argument --first: invalid int value: 'aa'
```

- 只要使用了 `--help` 选项，则忽略其他选项（即使其它选项有错误）

```
--help --first 1
```

输出：

```
usage: ./main [--help] [--first FIRST] [--second SECOND] [--third THIRD]

optional arguments:
--help   show this help message and exit
--first FIRST   First number
--second SECOND   Next number
--third THIRD   Third number
```

输入：

```
--fourth 4 --help
```

输出：

```
usage: ./main [--help] [--first FIRST] [--second SECOND] [--third THIRD]

optional arguments:
--help   show this help message and exit
--first FIRST   First number
--second SECOND   Next number
--third THIRD   Third number
```

提示

1. 在实现过程中，你可能需要进行较多的字符串处理。你可以对数组操作处理字符串，也可以使用 C++ STL 中的 `std::string` 进行更加方便的处理。具体使用方法可以查阅 https://cplusplus.com/reference/string/string/ (https://cplusplus.com/reference/string/string/) 。当然你也可以采用其他的具体处理方式。**注意：本题的评判不对具体实现作硬性要求。**
2. 在实现过程中，你可能需要实现字符串到整数和到字符串的映射。实现这种映射的方式有多种：（1）可以实现一个 `Argument` 类，维护加入到 `parser` 对象中的每个参数；（2）你也可以使用 C++ STL库中的 `std::map` ，通过 `std::map<std::string, int>` 与 `std::map<std::string, std::string>` 可以分别构造字符串到整数和到字符串的映射。具体使用方法可以查阅 http://www.cplusplus.com/reference/map/map/ (http://www.cplusplus.com/reference/map/map/) 。当然你也可以采用其他的具体实现方式。**注意：本题的评判不对具体实现作硬性要求。**
3. 在处理错误的过程中可能需要直接退出程序。可以直接使用 `exit(0)` （注意，不管是否出错都应该返回0，否则会被判定为 `Runtime Error` ）。

提交格式

- 你需要提交多个文件，包含Makefile，上述文件调用的各种头文件及其cpp文件；可以不包括提供的 `main.cpp` 文件。**Makefile必须要能生成可执行文件main（不带扩展名）**。
- 你应该将你的文件打包成一个zip压缩包并上传。**注意：你的文件应该在压缩包的根目录下，而不是压缩包的一个子文件夹下。**评测时，OJ会将提供的 `main.cpp` 贴入你的目录下进行编译并执行。

语言和编译选项

#	名称	编译器	额外参数	代码长度限制
0	custom	`make`		1048576 B

递交历史

#	状态	时间
表中没有数据		

递交答案

语言和编译选项 custom

```
1
```

提交

文件请拖入编辑器中，或 上传文件