Assignment 5 结构体、动态内存分配与链表

0.注意事项

cs1604库的引用

如果你想在这次作用中引入cs1604中的数据结构，请在source文件夹下新建cs1604.txt，将编译库产生的cs1604文件夹的**绝对路径**粘贴到cs1604.txt里，

上传时请包含这个txt，脚本仅会根据你是否有这个文件来判断是否引入StanfordcppLib，无需担心路径问题。

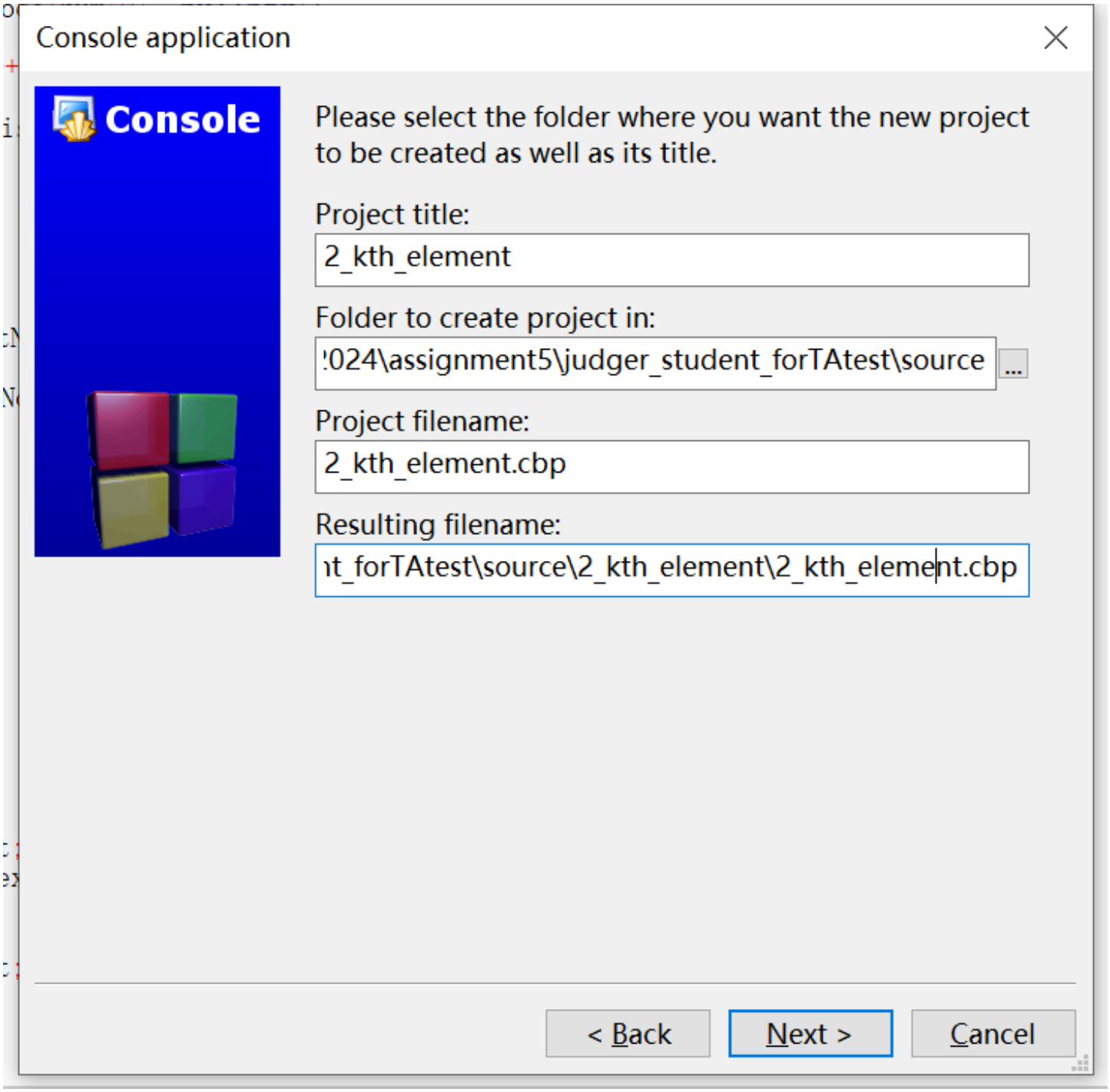
具体说明也可见source/README.md。

多文件编译

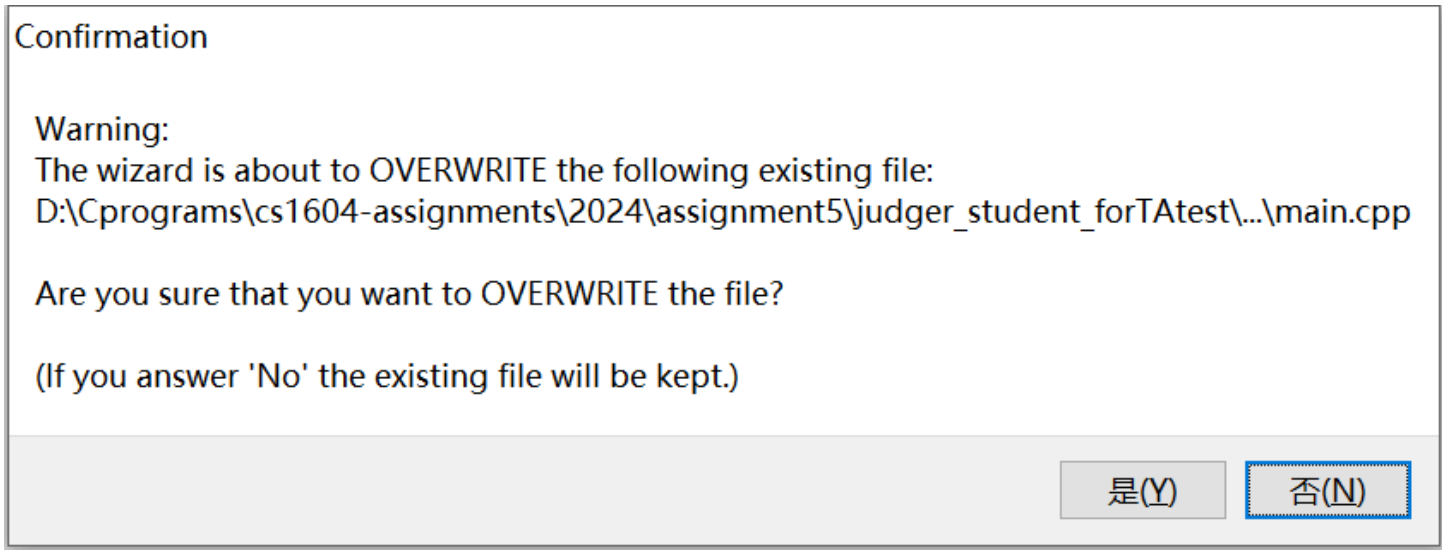
本次作业我们使用到了多文件编译，这里给出一些简单教程，当然，方案不唯一，你也可以上网去找一些相关说明。

CodeBlocks

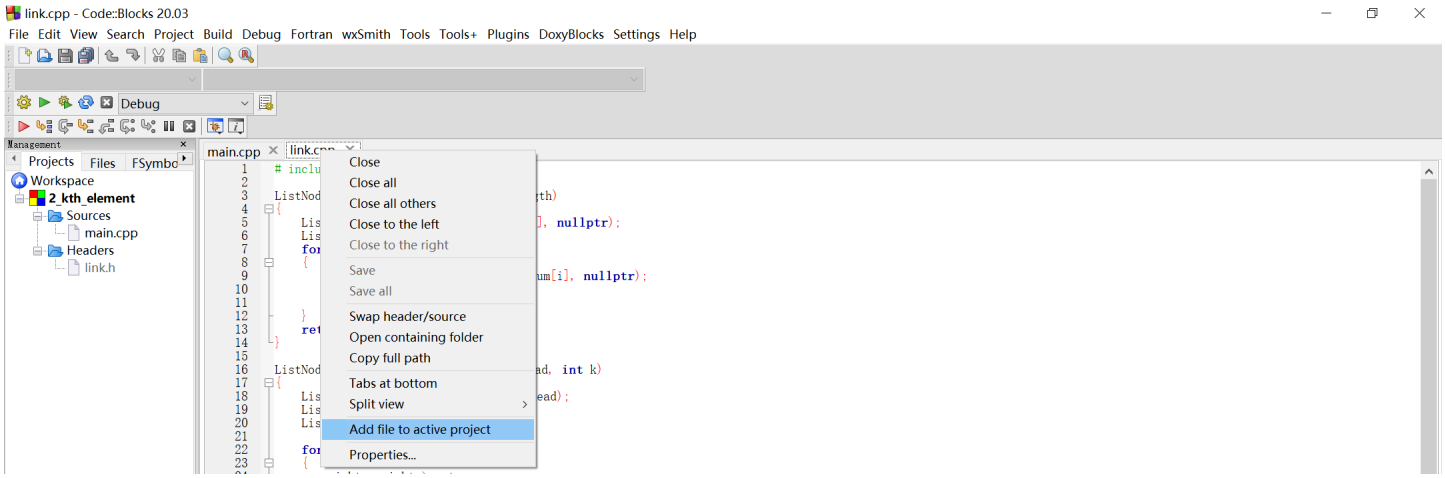
在CodeBlocks下你可以考虑在作业根目录下新建一个同名project（如下图所示）



选择"否"不覆盖现有文件。



然后将文件逐个打开并以如图所示的方式加入project，（选择默认ok选项即可）



最后build and run。

记得最后提交的时候把多余的文件删掉哦。

VScode

使用默认配置的同学可以尝试把tasks.json中args中的

```
"-g",
"${file}",
```

改为

```
"-g",
"${fileDirname}\\*.cpp",
```

使用code runner插件的同学也可以参考一下：

https://blog.csdn.net/IAMWisdomF/article/details/119705415

1.union

在学了struct这一章之后，小明一直有一个问题：union到底有什么用，为什么要把不同类型的数据放在同一块地方？于是助教打开了 `assignment3` 的 `asm_vm` 这一题，告诉他可以用一种新的视角来看待每一条输入的指令：每一条指令都可以作为一个特殊类型 `Struct T` 的一个元素，比如：

```
struct Command{
    enum type_name {A, B, C} type;
    union {
        int i;
        double d;
        char ch;
    } value;
};
```

这样，指令可以拆成两部分：操作符（Add，Sub，Assign等）是 `Command.type` ，可以使用switch枚举；操作数可以存放在 `Command.value` 中，且我们可以根据type方便地找值，不需要在输入时定义int，double和char来分别作处理。

小明若有所思，决定用这种新思路尝试一下这道题的修改版。

现在，你来和他一起尝试吧！

题目描述

在本题中，我们使用与上次作业类似的指令集，你需要实现如下结构：

模型

- PC：指向现在执行的语句，值为整数，从0开始
- state: 存储变量到其值的映射，变量的类型是 `char` ，值的类型是 `double`
- stack：存储操作数，类型为 `double`

指令

指令集在之前的基础上进行了拓展，请注意这些不同的地方：

- `AddL` ： 将栈顶两个元素出栈，相加并将结果入栈
- `AddR d` ： 取栈顶一个元素与立即数（或者说一个给定的右值）d相加，d的类型也是double；结果入栈
- `SubL` ： 取栈顶两个元素出栈，后出栈的减去先出栈的元素，结果入栈
- `SubR d` ： 同上，取栈顶一个元素减去d，结果入栈

> Mul和Div从原理上来说是完全一样的，~~由于这是最简单的第一题，~~ 为了给大家减轻负担，本题略去这两个指令。

- `Assign c`： 将栈顶元素d出栈，并更新state将变量名c映射到d，c的类型为 `char`
- `Var c`：将变量c的值放入栈顶
- `Jmp i`：跳转至pc=i处
- `JmpEq i`：将栈顶两个元素出栈，若两者相等，则跳转到pc=i处
- `JmpGt i`：将栈顶两个元素出栈，若第二个出栈的元素大于第一个出栈的元素，则跳转到pc=i处
- `JmpLt i`：小于关系，执行方式与 `JmpGt` 类似
- `Const d`：将类型为 `double` 的d置入栈顶
- `Print c`：输出变量c的值
- `Halt`：程序终止

文件描述

本题仅有一个main.cpp文件，其中关于结构体的定义已经给出，建议尝试在此基础上续写；或者你也可以定义更适配自己代码习惯的结构体。

输入输出

输入包含n+1行，其中第一行为一个整数n，表示接下来还要输入的行数；
接下来n行，每行为一条指令，**我们的pc地址从0开始计数。**

本题我们专注于结构体中的union，因此不考虑额外的错误问题；但在任何需要出栈操作时都需要进行检查（好习惯要养成），若出现栈为空但要出栈的情况，输出“Error”并**终止程序**

正常输出每次print为一行；为了保证答案一致匹配，对double的输出均保留两位小数，这一方法在 `assignment1` 中也提及过：
首先你需要在程序头部引入：

```
# include <iomanip>
```

之后在输出时使用如下格式的语句，例如，当你要输出ans保留两位小数的结果：

```
cout << fixed << setprecision(2) << ans << endl;
```

除了可能会有的空栈出栈，我们保证数据在其他部分合法。

例如：

输入

```
17
Const 10
Assign z
Const 0.5
Assign y
Var z
Const 0
JmpEq 15
Var y
Var z
AddL
Assign y
Var z
SubR 1
Assign z
Jmp 4
Print y
Halt
```

输出

```
55.50
```

解释
这个程序计算了0.5+1+2+3+... ...+10的结果。
数据范围
对于100%的数据，0<n<100

提示
`enum` 类型可以使用switch去枚举

2.kth_element

在学习了链表之后，小明想实现一个单向链表，并通过遍历来删除第k个数。但他转念一想又觉得找到第k个数太简单了，只需要一个计数器；因此他要给自己上上强度，去找链表中的倒数第k个数并删除。

人菜瘾大的小明很快发现自己做不出来了，所以来向你求助。

题目描述

在本题中，你需要完成一个单向链表的构建；通过遍历搜索来找到倒数第k个节点（这里我们与日常生活中的定义保持一致，即最后一个数据被定义为倒数第一个，其之前那个数据被定义为倒数第二

个），并将其删除；最终，你还需要实现删除整个链表的回收操作。每个节点包含两个元素，一个整数（int）和一个指向下一节点的指针（T*）。

需要完成的函数均在 `link.h` 中给出了函数头和描述，你可以自行查看。

文件描述

在本题中，我们希望大家可以体验一下如何进行多文件的编程。我们已经给出了link.h中的链表和函数定义，你需要在 `link.cpp` 中实现他们。同时，本题**不允许**你对 `main.cpp` 和 `link.h` 进行任何修改；在作业中助教已经提供了这两个文件，可以用作你写代码后进行测试的工具，但上传作业时**仅需要**提交 `link.cpp` （详见最后一部分提交说明），测评时会另外引入这两个文件进行测试。

输入输出

由于main函数已经给出，大家不需要关心输入的问题，但为了便于测试这里给出简短的说明：

输入

输入共有两行，第一行为n k，表示链表元素个数为n和要删除倒数第k个节点；
第二行为n个数，依次表示链表中各节点需要储存的元素。

输出

输出共两行，第一行包含n个元素，为建立列表后遍历得到的结果；
第二行包含n-1个元素，为删除倒数第k个节点后的遍历结果。

以下给出一组例子：

输入

```
5 4
1 2 3 4 5
```

输出

```
1 2 3 4 5
1 3 4 5
```

解释

5，4，3，2，倒数第4个节点为2.

数据范围

我们保证k是合理的，也就是说，倒数第k个节点一定存在。
对于100%的数据，$k \le n < 10000$

提示

1.一种常用的构建单链表的技巧是添加一个哑节点（dummy node），它不存放任何有意义的数据，仅有 `next` 指针指向链表的头节点，试思考这么做有什么作用。
2.无论删除后续哪一个节点的操作都是类似的，但如果需要删除头节点，我们可以考虑每次特殊处理，但有没有什么技巧可以让删除其余节点和头节点的方式一致呢?
3.如果你引入了额外的指针或者指针数组，务必记得在使用完毕后进行delete操作

进阶挑战

小明对你轻松做出了这道题感到不服气，他声称他在网上看到了一种大神解法，只需要**遍历链表一次**就可以完成这个任务，聪明如你能做到这一点吗?
（注：本挑战不做硬性要求，不计分，但欢迎同学们积极思考）
（另外，思路不止一种哦）

3.memory_allocator

这一次小明对动态内存分配的问题产生了兴趣，于是助教给了他一个简易的内存模型：
一段长为S（地址从0到S-1）的内存可以看成一排小方格，每个小方格代表一个内存地址，有两种状态（被占用或者空闲）。

每一次操作，系统会按某种规则分配给一个任务若干个连续内存格（为了作区分，下文对单个的内存地址称为“**格**”，对由一串内存构成的一个整体称为“**块**”），或者释放某个任务占据的所有内存块（一个任务可以占据多个内存块，且不一定是连续的，但这意味着它通过多次操作获取了多块内存；详见下文给出的样例输入输出）

懒惰的小明只想看演示，他不想真的写代码。现在，他把这个任务丢给了你。

题目描述

在本题中，我们实现一段长为S（地址为0到S-1）的内存中的各种操作，你需要考虑以下内容：

分配：将一块连续的内存分配给某个给定的任务

释放：释放内存中所有被给定任务使用中的内存块

压缩：将内存中所有被占用的内存向地址小的方向平移，以使得任务之间没有空闲的内存格（也就是说，所有空闲内存格全部在最后一个被占用的内存块之后）

关于内存分配的规则，我们给出如下三种规则的描述：

First：找到能容纳下需求且内存地址最小的位置进行分配

Biggest：找到目前内存中最大的空闲块，从该空闲块的头（内存地址较小的一端）开始分配；如果这样的最大块有两个及以上，取内存地址较小的那一个。

Smallest：找到目前内存中能容纳需求的最小空闲块，从该空闲块的头（内存地址较小的一端）开始分配；如果这样的最小块有两个及以上，取内存地址较小的那一个。

文件描述

本题中只有一个main.cpp，且没有任何内容填充，大家可以自己实现上述功能。

输入输出描述

输入的第一行为依次三个整数：s，n，m。其中s表示内存的总长度，n表示任务的总数（它们的标号自0到n-1），m表示进行操作的次数。

接下来m行，每一行会进行一次操作，可能的操作包括：

`R a`：释放所有由任务a占据的内存块

`S`：进行压缩内存操作

`A1 a x y`：将从x地址开始长度为y的内存块分配给任务a，如果分配不成功（与其他已被占据的内存块产生冲突）则输出一个“error”并换行表示分配不成功，这一行操作相当于无效，但**程序依然会继续运行**。

`A2 a y c`：将长度为y的内存块分配给任务a，分配方式为c。可能的c取值仅有三个：‘f’表示以First规则进行分配，‘b’表示以Biggest规则进行分配，‘s’表示以Smallest规则进行分配。你需要考虑内存分配不成功的情况（如当前最大的内存空闲块也无法满足需求），操作同上。

`O`：输出当前的内存情况。你仅需要打印被占用的内存块的情况，输出格式为每行一块被占用的内存，包含三个整数 `x y a`，分别表示起始地址，占用长度，和占用的任务进程编号。

注意：即使两块相邻的内存块拥有相同的任务编号，你也不需要合并他们变成一块；压缩内存仅仅是消去了中间的空闲内存。这是因为对于同一任务其可能需要多块内存来执行不同的任务（或者起到相互隔离的作用等），随意合并可能会引发错误。

但当我们释放内存时，相邻的空闲块会合并起来，以便后续的分配。

由于这题看起来就比较难，我们下面会提供一组样例输入输出并给出整个过程的简明说明。

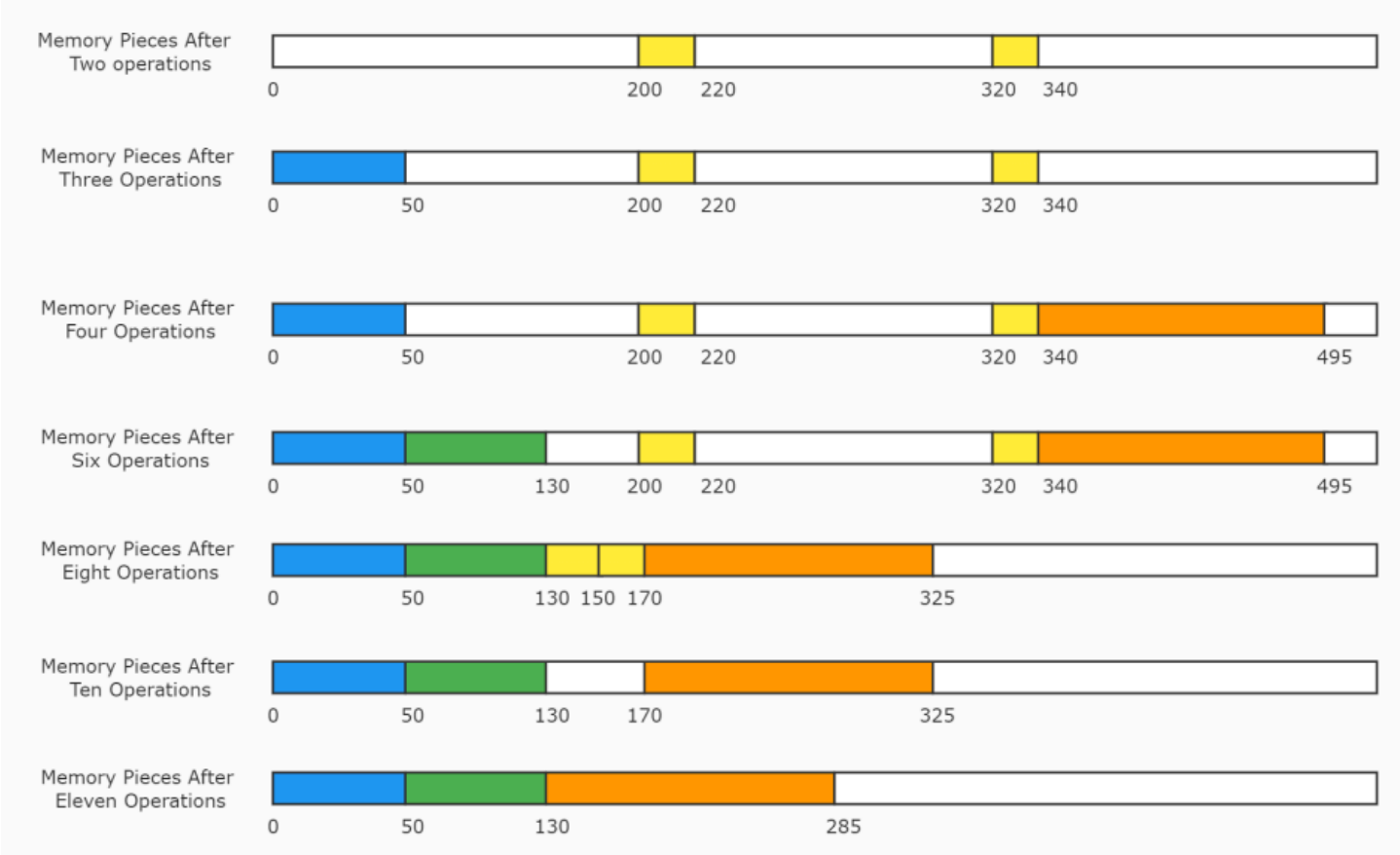
输入

```
500 4 12
A1 0 200 20
A1 0 320 20
A2 1 50 f
A2 2 155 s
A2 3 200 b
A2 3 80 b
O
S
O
R 0
S
O
```

输出

```
error
0 50 1
50 80 3
200 20 0
320 20 0
340 155 2
0 50 1
50 80 3
130 20 0
150 20 0
170 155 2
0 50 1
50 80 3
130 155 2
```

解释



一些关键步骤后的内存如图所示。

第三次操作使用First原则，因此[0，49]被分配给任务1。

第四次操作要求155长度的内存且使用Smallest原则，但仅有[340,494]还满足要求。

第五次操作请求200长度的内存，但当前空闲内存块的最大长度也仅有150，分配失败，输出error，此分配无效。

第六次操作成功，分配至[50，129]。

第七次操作输出了输出的第2行至第6行。

第八次操作进行了内存压缩，如图所示。（回想一下上文所说的注意）

第九次操作输出了输出的第7至第11行。

第十次操作释放了任务0占据的内存，如图黄色部分。

第十一次操作再度压缩内存。

第十二次操作输出了输出的第12行至14行。

数据范围

我们定义H为每个任务至多占据的内存块个数。

对于20%的数据，$n \le 100$；

对于30%的数据，$H \le 1$；

对于60%的数据，我们有$S \le 10^5$，$m \le 100$；

对于100%的数据，我们有$S \le 2^{31}-1$，$m \le 10^4$，$n \le 10^4$，$H \le 5$。

提示

本题直接维护一个长度为S的链表肯定是不大聪明的样子。但基于任务使用按块使用内存的特性，这里给出一种可行的思路：

链表中的每一个元素均是一块被占用的内存（思考一下至少需要保存哪些信息），初始时你可以引入第二题中已经介绍的哑节点来表示链表，之后每次分配均意味着按某种规则增加一个新节点；释放内存则是检测并删除若干节点。而压缩内存无需改变链表结构，仅需要改变某些特定节点中储存的信息即可。

另外，如果你使用其他类型的数据结构，请结合给出的测例输入考虑一下实现效率。

提交说明

你的提交文件格式应当为一个zip，zip压缩下的第一层是一个以你学号命名的文件夹，具体格式如下：

```
<your student number>.zip
|- <your student number>
|   |- 1_union
|   |  |- main.cpp
|   |
|   |- 2_kth_element
|   |  |- link.cpp
|   |
|   |- 3_memory_allocator
|   |  |- main.cpp
|   |
|   |- cs1604.txt（如果有）
```

说明：

1. 请不要提交多余的文件，保证格式正确！~~球球子助教不想再手工给有的同学调子。~~
2. 第二题只提交一个link.cpp，助教测评脚本时会额外引入一个main.cpp和link.h进行测试，是否提交main.cpp和link.h不影响最终结果，尝试通过修改这两个不变的文件来进行作弊是无效的，嘿嘿。