上 海 交 通 大 学 试 卷

（ 2022 至 2023 学年 第一学期 ）

班级号_____ 学号_____ 姓名 _____

课程名称 CS1953 - 程序设计（A 类） 成绩 _____

考前须知：

1. 本场考试形式为上机考试，考试时间 180 分钟，满分 300 分。在线监考平台为腾讯会议。考生需保证提前安装最新版本的软件并提前熟悉软件的使用（下载地址：https://meeting.tencent.com/download-center.html）。考场指令将在监视正面摄像头的腾讯会议发布，请不要将该会议静音。
2. 考试过程中，请遵守上海交通大学学生考试纪律规定。考试期间允许访问互联网、查看考试设备上存储的代码，允许查看任何形式的资料，但不允许任何形式的交流，不允许使用 copilot 等类似代码补全、代码编写插件或软件，一经发现，按作弊论处。
3. 如有任何疑问，请向**主持人私聊**。
4. 除卷面信息外，你需要在考试页面下载考试题目数据相关数据包，考试前 2 分钟将会在会议中公布数据包的密码。
5. 最终答案提交方法为提交源代码，各题目源代码文件名见下，所有题目只需要提交头文件中的内容。允许提交多次，每道题参与评分的代码为最后一次通过的提交，未通过的题目取最后一次得分最高的提交，但须在**考试时间内**完成提交。
6. 每道题同时会提供一个 cpp 文件，你可以编译该文件来测试运行自己的代码，题目描述中的输入格式和输出格式均指该文件，你也可以更改这个文件以方便调试自己的代码，但造成的后果由自己承担。

例如：对于第一题，在 main.cpp 文件内已经 #include "matrix.h"，你可以编译运行 main.cpp，并按照样例输入来测试自己完成的 matrix.h。特别需要注意的是此文件以及输入、输出等均只与调试有关，与最终测试无关。

7. 请严格按照题目描述的算法解决问题，否则将不能得到分数。
8. 内存泄漏扣分策略已在题面中写明，请注意检查空间的释放。
9. 请注意提交代码的代码风格！极为糟糕的代码风格（例如改变语法的某些宏定义）会酌情扣分！
10. 期末机考分数由人工审查获得。助教将通过综合评估程序的部分完成度、代码风格、内存泄漏、算法与代码提交是否合规等方面，并参考 OJ 评测结果，确定最终评分，极为糟糕的代码风格（例如改变语法的某些宏定义）亦会酌情扣分。对于不按照题目要求，错误实现或修改不可修改部分代码的，将严厉扣分。不保证最终成绩与 OJ 完成的结果相关。
11. 对于没有调试通过的成员函数，请将函数体与函数原型注释掉，以保证通过编译得到部分分。被注释的函数将在人工审查时适当给分。
12. 如遇到不可抗力因素，无法在考试时间内于线上评测系统提交的，请在 15 分钟内，将需要提交的代码发至 Canvas 平台的对应提交通道；如无法使用 Canvas 或仍无法正常提交的，请将代码以邮件的方式传至指定邮箱（邮箱地址：xia_tian@sjtu.edu.cn），标题为“期末机考代码补提交”，并在邮件正文中写明姓名、学号。如无法使用邮箱，请将代码通过即时通信软件（如微信、QQ 等）传送给老师或助教。超过 15 分钟未交的或不按要求提交代码的，视作未答题。

我承诺，我将严格遵守考试纪律。　　承诺人：_____

13. 评测与编译参数：

g++ code.cpp -o code -fmax-errors=10 -O2 -DONLINE_JUDGE -lm -std=c++17

gcc version: gcc version 12.2.0

题目时空限制与分值：

题目名称	A Toy MATLAB	A Naive Packet	MarkUp 文本编辑器
分值	100	100	100
源代码文件名	matrix.h	packet.h	markup.h
时间限制	6s	1s	6s
内存限制	512MiB	256MiB	256MiB

1 A Toy MATLAB

需要提交的文件：*matrix.h*。

Overview

> MATLAB 是美国 MathWorks 公司出品的商业数学软件，和 Mathematica、Maple 并称为三大数学软件。它在数学类科技应用软件中在数值计算方面首屈一指，广泛应用于矩阵运算、绘制函数和数据、实现算法、创建用户界面、连接其他编程语言的程序等。

矩阵是 MATLAB 中的基本元素。MATLAB 用由一对 `[]` 括起的若干数字表示一个矩阵：

```
[ 1 2 3 ; 4 5 6 ; 7 8 9 ]
```

其中行内元素以 `,` 或空格隔开，行与行间以 `;` 分隔。
你的任务是参照 MATLAB，实现一个带有基本矩阵运算功能的矩阵类。
简单来说，你需要实现一个带模板参数 `T` 的类 `Matrix`。

```
template <typename T> class Matrix;
```

表示一类元素类型为 `T` 的矩阵。具体地，它需要支持以下特性：

Features

含参构造

```
int seq[4] = {1, 2, 3, 4};
Matrix<int> a(2, 2, seq); // a := [ 1 2 ; 3 4 ]
Matrix<int> b(1, 2); // b := [ 0 0 ]
```

使用含参构造函数声明一个**已初始化**的矩阵实例。
第一个参数 `h` 表示矩阵的高（行数），第二个参数 `w` 表示矩阵的宽（列数）。**保证** $h, w > 0$ **且** $h \cdot w \leq 10^6$。
第三个参数为初始化序列，序列的第 $i \times w + j$（**0-base**）项表示矩阵第 `i`（**0-base**）行第 `j`（**0-base**）列的元素的初始值。
初始化序列**可省缺**，省缺时表示用元素类型**默认值**（即 `T()`）填充矩阵。

拷贝构造

```
Matrix<int> c(a); // c := [ 1 2 ; 3 4 ]
```

使用拷贝构造函数复制一个相同类型矩阵实例。
矩阵类型 `T` 不匹配时拷贝构造是**非法的**（保证不会在测试中出现）。

矩阵赋值

```
b = b; // b := [ 0 0 ]
c = b; // c := [ 0 0 ]
```

使用赋值运算将等号右端的对象值赋给等式左端对象。不同形状矩阵间的赋值是**合法的**。
矩阵类型 `T` 不匹配时赋值运算是**非法的**（保证不会在测试中出现）。

元素访问

```
Matrix<int> tmp1(a); // tmp1 := [ 1 2 ; 3 4 ]
const Matrix<int> tmp2(a); // tmp2 := [ 1 2 ; 3 4 ]

int ans1 = tmp2(1, 2); // ans1 := 2
tmp1(2, 2) = 1; // tmp1 := [ 1 2 ; 3 1 ]
```

使用 **1-base** 下标访问矩阵中的某一元素，对于非常量矩阵该访问应**可写**。
当访问的下标非法（例如 `ans1(0, 2)` 或 `ans1(1, 3)`）时，应立即停止运算，抛出 `MatrixIndexingError` 类型异常。

矩阵加法

```
Matrix<int> tmp3(1, 4, seq); // tmp3 := [ 1 2 3 4 ]
Matrix<int> tmp4(4, 1, seq); // tmp4 := [ 1 ; 2 ; 3 ; 4 ]

Matrix<int> ans2 = a + a; // ans2 := [ 2 4 ; 6 8 ]
Matrix<int> ans3 = tmp3 + tmp4;
// ans3 := [ 2 3 4 5 ; 3 4 5 6 ; 4 5 6 7 ; 5 6 7 8 ], implicit expansion
```

对于**形状相同**的矩阵，矩阵加法定义为

$$C_{n\times m} := A_{n\times m} + B_{n\times m} \iff c_{ij} = a_{ij} + b_{ij} \quad (0 \le i < n, 0 \le j < m)$$

特别地，对于**行向量 + 列向量**（`[1 x n] + [m x 1]` 或 `[m x 1] + [1 x n]`，这里 `[a x b]` 表示一个 `a` 行 `b` 列的矩阵）型矩阵加法，尽管矩阵形状不同，可以通过分别在行、列方向上重复补齐为 `[m x n]` 的矩阵，使矩阵加法有意义。例如：

$$\begin{bmatrix} a & b & c \end{bmatrix} + \begin{bmatrix} d \\ e \end{bmatrix} \Rightarrow \begin{bmatrix} a & b & c \\ a & b & c \end{bmatrix} + \begin{bmatrix} d & d & d \\ e & e & e \end{bmatrix} = \begin{bmatrix} a+d & b+d & c+d \\ a+e & b+e & c+e \end{bmatrix}$$

这一特性在 MATLAB 中被称为 **Implicit Expansion**。
若矩阵形状不相同且不满足上述 Implicit Expansion 条件，应立即停止运算，并抛出 `MatrixSizeError` 异常。
参与运算的矩阵类型不匹配时加法运算是**非法的**（保证不会在测试中出现）。

矩阵乘法

```
Matrix<int> ans4 = a * a; // ans4 := [ 7 10 ; 15 22 ]
Matrix<int> ans5 = tmp3 * tmp4; // ans5 := [ 30 ]
```

矩阵乘法定义为

$$C_{n\times s} := A_{n\times m} \times B_{m\times s} \iff c_{ij} = \sum_{k=0}^{m-1} a_{ik} \times b_{kj} \quad (0 \le i < n, 0 \le j < s)$$

当相乘的两个矩阵对应维大小不匹配时，应立即停止运算，抛出 `MatrixSizeError` 类型异常。
参与运算的矩阵类型 `T` 不匹配时乘法运算是**非法的**（保证不会在测试中出现）。

关于模板参数

保证类型 `T` 有定义默认构造函数、拷贝构造函数、析构函数、`operator=`、`operator==`、`operator+`（加法）、`operator*`（乘法）。
保证在正确使用类型 `T` 时不会抛出异常、内存泄漏、产生非法或未定义行为。

Structure

请补全以下代码实现 `Matrix` 类。
你可以选择自己喜欢的方式来存储矩阵元素值信息，例如：

- 使用二维数组指针：`T** mat`
- 使用嵌套的 `std::vector` 容器：`std::vector<std::vector<T>> mat`
- 将矩阵元素排列为一维序列，并使用一维数组指针 `T* arr` 或 `std::vector<T> arr` 存储
- 其他任何你喜欢的方式

特别地，以下存储方式不被接受：

- 使用固定大小的一维或二维数组
- 任何将不可避免地导致内存泄漏的实现

```
#ifndef MATRIX_H_
#define MATRIX_H_

#include <exception>
// just include whatever you want

namespace sjtu {

class MatrixSizeError : public std::exception {
```

```
 public:
  const char* what() noexcept {
    return "matrix size mismatch";
  }
};
class MatrixIndexingError : public std::exception {
 public:
  const char* what() noexcept {
    return "invalid matrix indexing";
  }
};

template <typename T>
class Matrix {
 private:
  int h, w;
  // TODO: choose your way to store elements in the matrix

 public:
  int height() const {
    return h;
  }
  int width() const {
    return w;
  }

  Matrix() = delete;
  Matrix(int h_, int w_, T* seq = nullptr) {
    // TODO
  }
  Matrix(const Matrix& o) {
    // TODO
  }

  Matrix& operator= (const Matrix& rhs) {
    // TODO
  }

  T& operator() (int row, int col) {
    // TODO
  }
```

```
  const T& operator() (int row, int col) const {
    // TODO
  }

  Matrix operator* (const Matrix& rhs) const {
    // TODO
  }

  Matrix operator+ (const Matrix& rhs) const {
    // TODO
  }

  ~Matrix() {
    // TODO
  }
};

} // namespace sjtu

#endif // MATRIX_H_
```

Samples

见下发数据包。

Testcases

本题对矩阵运算效率无硬性要求，部分测试点有内存泄漏检查。
保证运算过程中产生的所有矩阵大小（$h \cdot w$）不超过 10^6。

测试点编号	主要考查内容	内存泄漏检查	分值
1	样例	无	5
2	构造析构	无	10
3	矩阵赋值、元素访问	无	10
4	矩阵运算（不含 Implicit Expansion、异常处理、const）	无	10
5	矩阵运算（含 Implicit Expansion、异常处理、const）	无	15
6	鲁棒性测试	无	10
7	综合测试	无	20
8–14	1–7 号测试点	有	20

2 A Naive Packet

需要提交的文件：*packet.h*。

背景

> IP 是 **互联网协议**（internet protocol）的简称，是 TCP/IP 协议栈中的网络层协议。IP 协议在发展的过程中，衍生出 IPv4 和 IPv6 两个不同版本。其中，历史版本 IPv4 目前仍广泛使用；后继版本 IPv6 世界各地正在积极部署。
>
> IP 协议的通信单元是 **IP 包**（packet），同样分为 IPv4 和 IPv6 两个版本。它应该分为头部（header）和数据两大部分；其中头部应该包含源地址、目的地址以及数据类型等字段。

题目描述

> IPv4 地址由 32 位二进制数组成。为了方便记忆，将 IPv4 地址 32 位进制数进行分段，每段 8 位，共 4 段，然后将每段 8 位二进制数转换为十进制数，中间用 “.” 分割。这种表达方式称之为 **点分十进制**。
>
> IPv6 地址由 128 位二进制数组成，采用十六进制表示法，分 8 组表示，每组 16 位二进制数换算为 4 位十六进制数。各组之间用 “:” 号隔开，例如：1080:0000:0000:0000:0008:0800:200c:417a。
>
> 定义 0000 为零组。在 IPv6 地址段中有时会出现连续 n 组零组（$n \geq 1$），为了简化书写，这些 0 可以用 “::” 代替，但一个地址中只能出现一次 “::”。例如，上面的地址可以被简化为 1080::0008:0800:200c:417a。

现在，助教定义了 IP 地址的最简形式以及标准形式。

- 最简形式：
  - 对于 IPv4 地址，其处于最简形式当且仅当每段十进制数不含前导零。如：`127.0.0.1`。
  - 对于 IPv6 地址，其处于最简形式当且仅当每段十六进制数都由 4 位十六进制数表示，且不存在零组。如：`abcd:00a1::`。
- 标准形式：
  - 对于 IPv4 地址，其处于标准形式当且仅当每段十进制数都由 3 位十进制数表示。如：`127.000.000.001`。
  - 对于 IPv6 地址，其处于标准形式当且仅当每段十六进制数都由 4 位十六进制数表示，且不含 `::`。如：`abcd:0000:0000:0000:0000:0000:0000:0000`。

注意，十六进制数中大于 9 的数均应使用小写字母表示。

你的任务是实现一个简易的 Packet，示意图如图 1。

你需要完成：

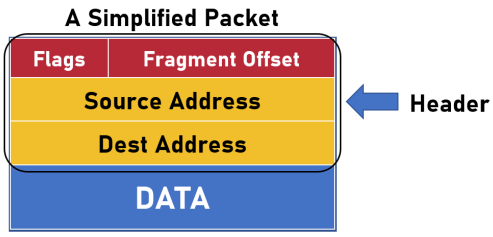

图 1: A Simplified Packet

构造函数与析构函数

你需要完成 `Packet`, `IPv4Packet`, `IPv6Packet` 类的构造函数与析构函数。构造函数参数共三个，类型均为 `const char*`，按顺序传入：

- **最简形式**的源地址 `src`，
- **最简形式**的目标地址 `dest`，
- 数据内容 `data`。

保证此三个参数对应的字符串在评测程序中不会被修改。

IP 包的分片函数

你需要完成 `IPv4Packet`，`IPv6Packet` 类的 `Segment(int MTU)` 函数。

> MTU (Maximum Transmission Unit) 是指网络支持的最大传输单元，以 **字节**为单位。MTU 的大小决定了发送端一次能够发送报文的最大字节数。
> IP 包的分片 (fragmentation) 是指将大的 IP 数据包分割成更小的单位。如果 IP 包的数据长度超过了 MTU，就要进行分片，使分片后的每个片段大小不超过 MTU。
> IPv4 允许路由器对超大的 IP 数据包进行分片，但 IPv6 不允许路由器执行分片操作。

对于 `IPv6Packet`，你 **只需要** 输出 `IPv6Packet is already segmented.\n`。
对于 `IPv4Packet`，你需要将 `char* data` 分为若干片，每片按如下格式输出：

```
Fragment <fragment>: Size <size>; Header Size <header_size>; Data Size <data_size>;
Flag <flags>; Fragment Offset <offset>; Data <segmented_data>.\n
```

- `<fragment>`：该分段的序号（从 0 开始）。
- `<size>`：该分段的大小（等于 `head_size` 与 `data_size` 的和）。
- `<header_size>`：头部的大小，默认为 20。
- `<data_size>`：`<segmented_data>` 的长度。
- `<flags>`：表示该分段是否为最后一段是否为中间段，中间段用 1 表示，最后一段用 0 表示。
- `<offset>`：表示该段在原数据中的偏移量，以 **8 字节**为单位。简单来说，设原报文为 `char* data`，那么该分段起始地址为 $data + offset \times 8$。因此，**除最后一段分段外**，所有分段的 `data_size` 应为 8 的倍数。为最大化利用一次能够传输报文的长度，除最后一段报文外，你应该取 `data_size` 为 **最大**的可能值。
- `<segmented_data>`：该分段所包含的数据，为字符串。

本题中 header size 均为 20。保证传入的 MTU 合法。
下面举一个具体的例子解释分段过程。对于以下的代码：

```
Packet* p = new IPv4Packet("192.168.0.1", "192.168.0.1", "00000000001111111111");
p->Segment(30);
```

MTU 为 30，减去 header 占用的 20，得出 data size 不超过 10，无法容纳长度为 20 的 data，故需要分段。
除最后一段分段外 data size 应为 8 的倍数，故在此例中，受 MTU 限制，data size 的最大可能值为 8。
由上述可知，data 共被分为三段，长度分别为 8，8，4。
对于前两段，为中间段，故 flag 为 1；对于最后一段，非中间段，故 Flag 为 0。
Offset 分别为每一段起始地址除以 8 的结果，即：

- `0 / 8 = 0`;
- `8 / 8 = 1`;
- `16 / 8 = 2`。

Data 为每一段分段所包含的内容。

- 对于第一段，起始地址为 0，长度为 8，故包含内容为 `00000000`。
- 对于第二段，起始地址为 8，长度为 8，故包含内容为 `00111111`。
- 对于第三段，起始地址为 16，长度为 4，故包含内容为 `1111`。

因此，输出结果应该是：

```
Fragment 0: Size 28; Header Size 20; Data Size 8; Flag 1; Fragment Offset 0; Data 00000000.
Fragment 1: Size 28; Header Size 20; Data Size 8; Flag 1; Fragment Offset 1; Data 00111111.
Fragment 2: Size 24; Header Size 20; Data Size 4; Flag 0; Fragment Offset 2; Data 1111.
```

打印函数

你需要完成 `IPv4Packet`，`IPv6Packet` 类的 `Print()` 函数。

- 首先，你应该调用 `Packet` 类的 `Print()` 函数；
- 其次，你应该分别输出（**标准形式**的 `src`，`dest`）以及 `data`，中间以 `'\t'` 作为分隔符；
- 最后，你应该输出换行符 `'\n'`。

提示 你可能需要以下工具帮助你完成本题。你可以点击以下超链接来访问这些网站。
注：如果选用 C 风格处理方式，注意在进行 `new` 操作时需要多 `new` 出一位以保存 `'\0'`，否则在调用下列 C-style 函数时可能会出现 Invalid read/write 的问题，从而导致 memory leak。

- C-style
  - strlen, strnlen_s - cppreference.com
  - strcat, strcat_s - cppreference.com
  - strcpy, strcpy_s - cppreference.com
  - strtok, strtok_s - cppreference.com

  - memset, memset_explicit, memset_s - cppreference.com
  - putchar - cppreference.com
- std::string
  - std::basic_string - cppreference.com
- std::stringstream
  - std::basic_stringstream - cppreference.com

你需要使用以下模板完成此题。你不应该修改模板已经存在的任何内容（标注内容除外）；你不应该包含其他头文件；你不应该新增 `Packet` 类的成员变量。保证所有输入均合法。

```
#ifndef PACKET_H_
#define PACKET_H_

#include <iostream>
#include <sstream>
#include <cstring>
#include <vector>

class Packet {
 private:
  // You should let ... be whatever you want.
  using T = ...;
  T src;
  T dest;
  char* data;

 public:
  // TODO: constructor and destructor
  const T& GetSrc() const { return src; }
  const T& GetDest() const { return dest; }
  const char* GetData() const { return data; }
  virtual void Segment(int MTU) = 0;
  virtual void Print() const {
    std::cout << "Print start..." << std::endl;
  }
};

class IPv4Packet : public Packet {
 private:
  // Do whatever you want
 public:
```

```
  // TODO
};

class IPv6Packet : public Packet {
 private:
  // Do whatever you want
 public:
  // TODO
};

#endif // PACKET_H_
```

样例测试程序 1

```
#include "packet.h"
int main() {
  Packet* p = nullptr;
  p = new IPv4Packet("192.168.0.1", "192.168.0.1", "00000001111111");
  p->Print();
  delete p;
  p = new IPv6Packet("1080::0008:0800:200c:417a", "1080::0008:0800:200c:417a",
    "00000001111111");
  p->Print();
  delete p;
  return 0;
}
```

样例测试程序 2

```
#include "packet.h"
int main() {
  Packet* p = nullptr;
  p = new IPv4Packet("192.168.0.1", "192.168.0.1", "0000000001111111111");
  p->Segment(30);
  delete p;
  p = new IPv6Packet("1080::0008:0800:200c:417a", "1080::0008:0800:200c:417a",
    "0000000001111111111");
  p->Segment(30);
  delete p;
  return 0;
}
```

样例输出 1

```
Print start...
192.168.000.001        192.168.000.001        00000001111111
Print start...
1080:0000:0000:0000:0008:0800:200c:417a        1080:0000:0000:0000:0008:0800:200c:417a
00000001111111
```

样例输出 2

```
Fragment 0: Size 28; Header Size 20; Data Size 8; Flag 1; Fragment Offset 0; Data 00000000.
Fragment 1: Size 28; Header Size 20; Data Size 8; Flag 1; Fragment Offset 1; Data 00111111.
Fragment 2: Size 24; Header Size 20; Data Size 4; Flag 0; Fragment Offset 2; Data 1111.
IPv6Packet is already segmented.
```

测试点说明

测试点	测试函数	内存泄漏检查	分数
1	Test for `IPv4Packet: Print`	无	8
2	Test for `IPv6Packet: Print`	无	12
3	Harder test for `IPv6Packet: Print`	无	12
4	Test for: `Segment`	无	24
5	Test for: `Segment`	无	8
6	All	无	8
7	All	无	8
8–14	同 1–7	有	20

3 MarkUp 文本编辑器

需要提交的文件：*markup.h*。

你的任务是完成类 `MarkUp`，这是一个每个元素是一个 `char` 的链表的链表实现的文本编辑器。

这个文本编辑器可以被理解为二维不定长度数组，但是我们用链表套链表来实现。具体来说，每一行是一个把这一行所有字符串起来的一个链表，然后我们把每一行的链表头连起来，又成为一个链表。其中每个 `node` 我们储存一个不为 `\n` 的 `char`。具体结构可为：

```
headNode_headPointer
     |(nextHead)
headNode_row_1 -> (rowHead) Node('A') -> Node('C') -> Node('M') -> ...
     |(nextHead)
headNode_row_2 -> (rowHead) Node('2') -> Node('2') -> ...
     |(nextHead)
    ...
```

例如以上结构所指代的文本为：

> ACM
> 22

本题应该实现的函数包括：

#	函数	解释
-1	`print()`	先输出“==========MARK_UP_OUTPUT==========\n”，然后打印文本
0	constructor, destructor	构造函数、析构函数
1	`typeIn(ch)`	在全文最末尾打入字符 `ch`
2	`insert(row, col, ch)`	在第 `row` 行, 第 `col` 个位置插入字符 `ch`
3	`erase(row, col)`	删除第 `row` 行, 第 `col` 个字符
4	`query(row, col)`	返回第 `row` 行, 第 `col` 个字符
5	`split(row, col)`	相当于在 `row` 行, 第 `col` 个空位置插入换行符，该行裂成两行
6	`copy_paste(a1, a2, b1, b2, c1, c2)`	将从【a1 行第 a2 个字符】到【b1 行第 b2 个字符】这段内容复制并插入到【c1 行第 c2 个位置】处

具体解释：

以下列文本为例：

> ABCDEFG
> HIJKLMN

`print`：

需要注意，文末的空行也要输出来。

`query`，`erase`：

一个字所在的位置我们用第几行第几个字的坐标来表示。所有的位置都是 1-based 的，例如：在上述文本中，

'A' 的位置是第一行第一个字（1，1），'I' 的位置是第二行第二个字（2，2）。

`insert`：

如果我们对其进行插入，则其插入的位置也都是 1-based 的。例如，在上述文本中在 'G' 字后面加 ','，插入的位置为（1，8），对应的函数调用为：`insert(1, 8, ',')`。测试中不会插入到新行，例如对以上文本执行 `insert(3, 1, 'O')` 的操作是不会出现的，`copy_paste` 的 paste 部分也满足该要求。

`typeIn`：

直接插入在全文最末。

`split`：

换行。例如对上述文本进行 `split(1, 3)`，将会得到：

> AB
> CDEFG
> HIJKLMN

- 若换行发生在边界，则会产生空行。例如，对上述文本进行 `split(1, 8)` 将会得到：

> ABCDEFG
> //（这是一个空行）
> HIJKLMN

- `split(1, 1)` 将得到：

> //（这是一个空行）
> ABCDEFG
> HIJKLMN

`copy_paste`:

需要注意复制的内容是包含两端的字的。

另外插入（粘贴）的规则与 `insert` 相同。为简单起见，测试点中的复制区域与粘贴区域**不会**重叠。

额外说明

1. 若不使用链表套链表结构，而是使用单层链表等不符合题目要求的实现方法通过本题，你在本题中将获得不超过 80% 的分数。
2. 保证所有测试点所有输入合法或无输入。

测试点说明

测试点	测试的函数	内存泄漏检查	分数
1	-1, 0, 1	无	6
2	-1, 0, 1	无	6
3	-1, 0, 1	无	6
4	-1, 0, 1, 2, 3, 4	无	6
5	-1, 0, 1, 2, 3, 4	无	6
6	-1, 0, 1, 2, 3, 4	无	6
7	-1, 0, 1, 2, 3, 4, 5	无	6
8	-1, 0, 1, 2, 3, 4, 5	无	6
9	-1, 0, 1, 2, 3, 4, 5	无	6
10	-1, 0, 1, 2, 3, 4, 5	无	6
11	All	无	6
12	All	无	6
13	All	无	6
14	All	无	6
15	All	无	6
16–20	同 11–15	有	10

样例程序（sample.cpp）

```cpp
// correctness: typeIn, insert, erase, split, query, copy_paste

#include <iostream>
#include "markup.h"

using namespace sjtu;

MarkUp mu;

int main() {
  mu.typeIn('A');
  mu.typeIn('S');
  mu.typeIn('O');
  mu.typeIn('U');
  mu.typeIn('L');

  for (int i = 1; i <= 5; i++) {
    std::cout << mu.query(1, i) << ' ';
  }
```

```
  std::cout << std::endl;
  // A S O U L


  mu.insert(1, 2, '-');
  mu.print();
  // A-SOUL

  mu.split(1, 7);
  mu.typeIn('1');
  mu.typeIn('-');
  mu.copy_paste(1, 1, 1, 6, 2, 3);

  mu.print();
  // A-SOUL
  // 1-A-SOUL

  mu.erase(2, 2);


  mu.print();
  // A-SOUL
  // 1A-SOUL

  mu.split(2, 1);
  mu.print();
  // A-SOUL
  //
  // 1A-SOUL

  return 0;
}
```

样例输出（sample.ans）

```
A S O U L
==========MARK_UP_OUTPUT==========
A-SOUL
==========MARK_UP_OUTPUT==========
A-SOUL
1-A-SOUL
```

```
==========MARK_UP_OUTPUT==========
A-SOUL
1A-SOUL
==========MARK_UP_OUTPUT==========
A-SOUL

1A-SOUL
```