****

**实验报告：银行业务**

2351041

刘浩田

完成日期:2024.9.28

# 索引

### 概述

### 项目目的

### 项目环境

### 项目背景

### 项目要求

### 项目实现

### 项目总结

# 概述

在现代社会中，银行业务处理是金融领域的重要组成部分，其效率和准确性直接影响顾客的满意度和银行的运营效率。随着信息技术的快速发展，银行业务处理逐渐从传统的人工操作转向电子化、自动化处理。本项目旨在设计并实现一个银行业务处理模拟系统，通过模拟银行中A、B两个业务窗口的工作流程，研究不同窗口处理速度差异对顾客服务体验的影响。

本系统的核心功能是接收一组顾客编号作为输入，根据编号的奇偶性将顾客分配到A或B窗口，并根据窗口的处理速度差异，模拟整个业务处理过程。最终，系统将按照业务完成的顺序输出顾客的编号。该系统不仅能够处理大量顾客数据，还能确保输出顺序的准确性和高效性。

在系统设计上，我们采用了面向对象的编程方法，将顾客信息抽象成对象，通过构建一系列类和数据结构来实现信息的组织和管理。系统的主要模块包括信息输入模块、信息处理模块、信息输出模块和用户交互界面。

信息输入模块负责接收用户输入的顾客总数和编号，系统会验证输入的合法性，防止无效或错误信息的录入。信息处理模块是系统的核心，它根据顾客编号的奇偶性将顾客分配到相应的窗口，并模拟业务处理过程。信息输出模块负责将处理完成的顾客编号按照业务完成的顺序输出。

用户交互界面为用户提供了友好的操作界面，通过控制台输入和输出，使用户能够方便地与系统进行交互。在用户输入操作指令时，系统会给出清晰的提示信息，指导用户完成操作。为了确保系统的健壮性，本实验还在系统实现过程中充分考虑了异常情况的处理，如输入验证、错误提示等。输出的顾客编号也经过了格式化处理，便于用户阅读和理解。

在技术实现上，本系统采用了C++语言进行编程，利用了其强大的面向对象编程特性。系统的核心数据结构是队列，每个队列节点存储了一个顾客的信息。队列的结构特点使得顾客信息的插入和删除操作非常高效。同时，我们也自定义了一些辅助类，如`checkCin`函数，用以处理字符串的输入输出，增强了系统的健壮性。

此外，系统还提供了友好的用户交互界面，通过控制台输入和输出，使用户能够方便地与系统进行交互。在用户输入操作指令时，系统会给出清晰的提示信息，指导用户完成操作。为了确保系统的健壮性，本实验还在系统实现过程中充分考虑了异常情况的处理，如输入验证、错误提示等。输出的顾客编号也经过了格式化处理，便于用户阅读和理解。

为了确保程序的广泛适用性，我们在设计时特别考虑了跨平台兼容性，使得该程序不仅在Windows操作系统上运行流畅，同时也兼容Linux环境，满足了不同用户群体的需求。通过使用标准C++语言和遵循Linux标准的函数，我们的程序能够在两大主流操作系统上提供一致的用户体验。

本项目旨在通过电子化手段优化银行业务处理流程，提高银行服务的效率和质量。通过本系统的实现，我们不仅能够提升顾客办理业务的便捷性，还能够减轻银行工作人员的工作负担，实现金融资源的优化配置。

# 项目目的

本项目的核心目的在于通过设计和实现一个银行业务处理模拟系统，深化学生对数据结构、算法以及面向对象编程的理解和应用能力。在计算机科学教育中，理论与实践相结合是培养学生解决实际问题能力的重要途径。通过本实验，学生不仅能够将课堂上学到的理论知识运用到具体项目中，还能够在实践中进一步提升自己的编程技巧和软件设计能力。

首先，实验旨在加强学生对数据结构的理解和应用。银行业务处理系统涉及到大量的数据处理，如顾客信息的存储、查询、修改和删除等，这些操作都需要高效的数据结构来支持。通过本实验，学生可以深入理解队列、类等数据结构的特点和适用场景，掌握它们在实际问题中的应用方法。特别是在处理动态数据时，队列结构的灵活性和高效性显得尤为重要。

其次，实验强调了算法的重要性。一个优秀的算法能够显著提高程序的执行效率，对于银行业务处理系统而言，这直接关系到用户体验。例如，在实现顾客信息查询功能时，选择合适的搜索算法可以加快检索速度；在进行数据统计时，合理的算法可以减少计算的复杂度。通过本实验，学生可以练习设计和优化算法，提高解决实际问题的效率。

再次，实验着重于提升学生的面向对象编程能力。面向对象编程是一种重要的编程范式，它通过将数据和操作封装成对象，提高了代码的可重用性和可维护性。在本实验中，学生需要构建多个类来表示不同的概念，如顾客、窗口等，并设计类的属性和方法来实现系统的功能。这要求学生不仅要掌握面向对象编程的语法，还要学会如何运用面向对象的思想进行系统设计。

此外，实验还旨在培养学生的软件工程意识。在实际的软件开发过程中，除了编写代码，还需要进行需求分析、系统设计、测试和维护等工作。通过本实验，学生可以体验软件开发的全过程，了解各个阶段的任务和要求。特别是在系统设计阶段，学生需要考虑系统的结构、模块划分、接口设计等问题，这对于提高学生的系统分析和设计能力具有重要意义。

最后，实验还关注于提升学生的用户体验意识。一个好的软件不仅要功能强大，还要易于使用。在本实验中，学生需要设计用户友好的交互界面，确保用户能够方便地进行操作。这要求学生不仅要关注程序的内部逻辑，还要考虑用户的需求和习惯，培养从用户的角度思考问题的能力。

综上所述，本实验目的在于通过实际操作，使学生在多方面得到锻炼和提升，为将来的学习和工作打下坚实的基础。通过本实验，学生不仅能够提高自己的编程能力和系统设计能力，还能够培养良好的软件开发习惯和用户意识，为成为一名优秀的软件工程师奠定基础。

# 项目环境

## 系统：Windows11

## 编译器：VisualStudio2022x86

## 语言：C++

# 项目背景

随着信息技术的不断发展，传统行业正经历着数字化转型，银行业务处理作为金融服务的重要组成部分，其效率和质量直接影响着顾客的满意度和银行的竞争力。传统的银行业务处理方式，如手工登记、排队等候等，不仅效率低下，而且容易出错，难以满足现代社会对金融服务的高效率和个性化需求。

在现代银行业务中，顾客到达的随机性和业务处理速度的差异性是常见的问题。为了提高服务效率和顾客满意度，需要合理分配顾客到不同窗口办理业务。本项目通过模拟银行中A、B两个业务窗口的工作流程，研究不同窗口处理速度差异对顾客服务体验的影响。

本项目旨在通过电子化手段优化银行业务处理流程，提高银行服务的效率和质量。通过本系统的实现，我们不仅能够提升顾客办理业务的便捷性，还能够减轻银行工作人员的工作负担，实现金融资源的优化配置。此外，本项目还能够帮助学生理解现实世界中的业务逻辑和操作流程，为将来的职业生涯打下坚实的基础。

在这样的背景下，开发一个功能完善、操作简便、安全可靠的银行业务处理模拟系统变得尤为重要。本项目正是在这样的需求驱动下进行的设计和实现。系统旨在为教育机构提供一个电子化的解决方案，以应对日益增长的业务需求，并解决传统业务处理方式中存在的问题。。

# 项目要求

本项目旨在开发一个银行业务处理模拟系统，该系统能够模拟银行中A、B两个业务窗口的工作流程，并考虑不同窗口处理业务的速度差异。系统需要能够根据顾客到达的顺序和业务需求，按照业务完成的顺序输出顾客的编号。以下是对项目的详细要求：

## 6.1. 功能要求

-1.输入顾客信息

能够通过用户输入的方式添加顾客信息。输入应包括顾客总数和各个顾客的编号，其中编号为奇数的顾客需要到A窗口办理业务，为偶数的顾客则去B窗口。

- 2.输出顾客信息

能够按照业务处理完成的顺序输出顾客的编号。输出的数字间以空格分隔，最后一个编号后不应有多余的空格。

- 3.业务处理模拟

系统需要模拟A、B两个窗口的业务处理过程，确保A窗口每处理完2个顾客时，B窗口处理完1个顾客。当不同窗口同时处理完2个顾客时，A窗口的顾客优先输出。

## 6.2. 数据结构要求

- 数据组织：建议使用队列来存储顾客信息，以支持高效的数据插入和删除操作。

- 数据完整性：确保所有顾客信息的完整性和准确性，如检查必填项、格式等。

## 6.3. 用户界面要求

- 操作简便：用户界面应直观易用，操作流程简单明了。

- 输入验证：系统应提供输入验证，确保用户输入的数据符合要求。

- 错误处理：对于用户的错误操作，系统应提供清晰的错误提示。

## 6.4. 性能要求

- 效率：系统应能够高效地处理大量顾客数据，确保输出顺序的准确性和高效性。

- 健壮性：系统应具备良好的健壮性，能够处理各种异常情况，如输入错误、数据格式错误等。

## 6.5. 可复用性和可维护性

- 模块化设计：系统应采用模块化设计，便于后期维护和升级。

- 代码可读性：代码应具有良好的可读性，注释清晰，便于他人理解和修改。

## 6.6. 测试要求

- 测试用例：为每个功能编写详细的测试用例，确保功能的正确性和稳定性。

- 边界测试：进行边界条件测试，确保系统在极端情况下也能正常工作。

通过满足以上要求，银行业务处理模拟系统将能够为用户提供一个功能全面、操作简便、安全可靠的银行业务模拟管理工具，有效提升银行业务处理的效率和质量。

# 项目实现

本项目使用C++语言实现，采用面向对象的编程方法，主要利用模板类和队列数据结构来模拟银行业务处理流程。项目的核心是模拟A、B两个窗口处理顾客业务的过程，其中A窗口的处理速度是B窗口的2倍。

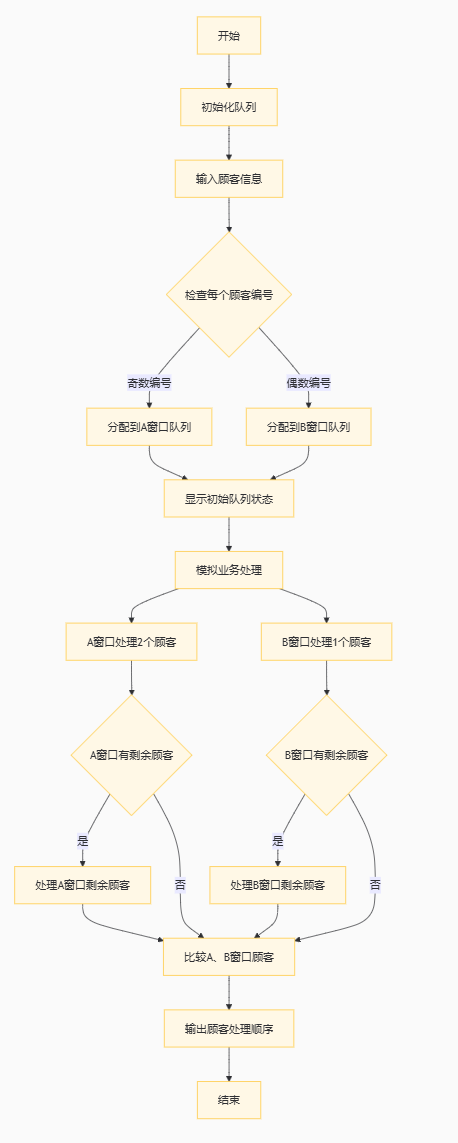


图 项目流程图

## 7.1 数据结构设计

项目的核心数据结构是模板类Queue<T>，它实现了一个循环队列。这种设计允许队列存储不同类型的数据，增加了代码的复用性和灵活性。队列类的主要组件包括：  
  
- 私有成员变量：  
front和rear：这两个整型变量用于跟踪队列的头部和尾部位置。在循环队列中，它们的值会在0到maxSize-1之间循环。  
elements：这是一个指向T类型的指针，用于动态分配数组来存储队列元素。使用动态分配允许在运行时确定队列大小，提高了内存使用的灵活性。  
maxSize：表示队列的最大容量，这个设计允许在创建队列时指定其大小，提高了内存使用效率。  
  
- 公有成员函数：  
这些函数实现了队列的基本操作，如入队（EnQueue）、出队（DeQueue）、获取队首元素（GetFront）、判空（IsEmpty）、判满（IsFull）等。这些操作保证了队列操作的正确性和效率。  
  
特别值得注意的是CheckQuene函数，它实现了根据元素的奇偶性来决定该元素应该进入哪个队列的逻辑。这个函数直接体现了题目要求中的顾客分配规则。

|  |
| --- |
| 附件、Queue类 |
| class Queue {  public:  Queue(int size = MAX\_SIZE) : maxSize(size), front(0), rear(0) {  elements = new T[maxSize];  assert(elements != nullptr);  }  ~Queue()  void EnQueue(const T& item)  T DeQueue()  T GetFront()  void MakeEmpty()  bool IsEmpty()  bool IsFull()  int Length()  int CheckQuene(const T& item)  void Print()  private:  int front, rear;  T\* elements;  int maxSize;  }; |

## 7.2 主要算法实现

主程序的实现采用了分步骤的方法，确保了程序的清晰度和可维护性。以下是每个步骤的详细解释：  
  
1. 输入处理：  
使用专门的输入处理函数checkCin来读取用户输入。这个函数的设计增强了程序的健壮性，它不仅检查输入是否为有效数字，还确保输入在指定范围内。例如，在读取顾客总数N时，它会确保N不超过1000。这种设计大大提高了程序的用户友好性和错误处理能力。  
  
2. 顾客分配：  
程序创建了两个Queue对象Q\_A和Q\_B，分别代表A窗口和B窗口的顾客队列。然后，程序遍历输入的顾客编号，使用CheckQuene函数判断每个顾客应该进入哪个队列。这个过程直接实现了题目要求中的顾客分配规则：编号为奇数的顾客进入A窗口，偶数的进入B窗口。  
  
3. 显示初始队列状态：  
在分配完成后，程序会打印出A窗口和B窗口的初始顾客队列。这一步虽然不是题目的直接要求，但它提供了中间结果，方便用户理解和验证程序的正确性。  
  
4. 业务处理模拟：  
这是整个程序的核心部分，也是最复杂的部分。程序创建了一个新的队列Q\_out来存储处理完成的顾客顺序。处理过程分为以下几个步骤：  
  
**a. 计算同时处理的次数**

程序计算出一个time\_1变量，它是A窗口队列长度的一半和B窗口队列长度中的较小值。这个计算基于A窗口处理速度是B窗口2倍的规则。  
  
**b. 同时处理阶段**

在这个阶段，程序模拟了A窗口处理两个顾客，B窗口处理一个顾客的过程。具体来说，它会循环time\_1次，每次从A窗口队列中取出两个顾客，从B窗口队列中取出一个顾客，按照这个顺序将它们加入到Q\_out队列中。  
  
**c. 处理剩余顾客**

在同时处理阶段结束后，可能还有顾客未被处理。程序首先检查A窗口是否还有顾客，如果有，先处理一个，然后再处理所有剩余的。这个设计考虑到了A窗口可能会多出一个顾客的情况。如果A窗口为空但B窗口还有顾客，则处理B窗口的所有剩余顾客。  
  
这个算法巧妙地实现了A窗口处理速度是B窗口2倍的要求，同时也处理了当窗口同时处理完顾客时A窗口优先的规则。

|  |
| --- |
| 附件、业务模拟部分 |
| Queue<int> Q\_out(1000);  int time\_1;  if (Q\_A.Length() / 2 > Q\_B.Length()) {  time\_1 = Q\_B.Length();  }  else {  time\_1 = Q\_A.Length() / 2;  }  for (int i = 0; i < time\_1; i++) {  Q\_out.EnQueue(Q\_A.GetFront());  Q\_A.DeQueue();  Q\_out.EnQueue(Q\_A.GetFront());  Q\_A.DeQueue();  Q\_out.EnQueue(Q\_B.GetFront());  Q\_B.DeQueue();  }  if (!Q\_A.IsEmpty()) {  Q\_out.EnQueue(Q\_A.GetFront());  Q\_A.DeQueue();  }  if (!Q\_A.IsEmpty()) {  for (; Q\_A.Length(); ) {  Q\_out.EnQueue(Q\_A.GetFront());  Q\_A.DeQueue();  }  }  else if (!Q\_B.IsEmpty()) {  for (; Q\_B.Length(); ) {  Q\_out.EnQueue(Q\_B.GetFront());  Q\_B.DeQueue();  }  } |

5. 输出结果：  
最后，程序打印出Q\_out队列中的顾客序列，这就是最终的顾客处理顺序。

## 7.3 关键函数实现

1. Queue<T>::CheckQuene 函数：  
这个函数实现了顾客分配的核心逻辑。它接受一个顾客编号作为参数，然后判断这个编号的奇偶性。如果是偶数，函数返回0，表示顾客应该进入B窗口；如果是奇数，函数返回1，表示顾客应该进入A窗口。这个简单的判断直接体现了题目的关键要求。

|  |
| --- |
| 附件、CheckQuene 函数 |
| int CheckQuene(const T& item) {  if (item % 2 == 0) {  return 0;  }  else {  return 1;  }  } |

2. checkCin 函数：  
这个函数加强了输入的错误处理能力。它的实现采用了一个无限循环，直到获得有效输入才退出。在循环中，它首先尝试读取一个整数值。如果读取失败（例如，用户输入了非数字字符），它会清除错误标志，忽略无效输入，并提示用户重新输入。如果成功读取了一个整数，它会检查这个整数是否在指定的范围内。如果不在范围内，同样会提示用户重新输入。只有当读取到一个在有效范围内的整数时，函数才会结束循环并返回这个值。这个函数的设计大大提高了程序的用户友好性和健壮性，防止了可能由于错误输入导致的程序崩溃。

|  |
| --- |
| 附件、checkCin 函数 |
| void checkCin(int& value, const int lower\_limit = 0, const int higher\_limit = numeric\_limits<int>::max(), const int erase\_puff\_area = 1)  {  while (1) {  cin >> value;  if (cin.fail()) {  cout << "输入非法，请重输\n";  cin.clear();  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  continue;  }  else if (value < lower\_limit || value > higher\_limit) {  cout << "输入超限，请重输\n";  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  continue;  }  else {  if (erase\_puff\_area) {  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  }  break;  }  }  } |

## 7.4 主函数实现

主函数main()通过有序的步骤实现了整个银行业务处理的流程。它首先使用checkCin函数读取顾客总数和每个顾客的编号，确保了输入数据的正确性。然后，它使用两个Queue对象来模拟A和B窗口的顾客队列，并使用CheckQuene函数来决定每个顾客应该进入哪个队列。  
  
在顾客分配完成后，主函数会显示初始的队列状态，这增加了程序的可视化效果，有助于用户理解程序的运行过程。接下来，它进入了业务处理的核心逻辑，使用一个新的Queue对象Q\_out来存储处理完成的顾客顺序。  
  
处理过程中，主函数巧妙地实现了A窗口处理速度是B窗口2倍的要求。它首先计算出可以同时处理的次数，然后在每次处理中，从A窗口取出两个顾客，从B窗口取出一个顾客。这个过程重复进行，直到其中一个窗口的顾客处理完毕。然后，它会处理剩余的顾客，确保所有顾客都得到服务。  
  
最后，主函数会打印出Q\_out队列中的顾客序列，这就是最终的顾客处理顺序，直观地展示了模拟结果。  
  
通过这种实现方式，程序成功模拟了银行两个窗口的业务处理过程，并按照要求输出了顾客的处理顺序。程序不仅考虑了A窗口处理速度是B窗口2倍的情况，还处理了当窗口同时处理完顾客时A窗口优先的规则，充分体现了问题的所有要求。  
  
整个实现过程展现了面向对象编程的优势，通过合理的类设计和函数封装，使得程序结构清晰，易于理解和维护。同时，使用模板类的设计也为将来可能的扩展留下了空间，增强了程序的可扩展性。例如，如果将来需要处理不同类型的顾客（比如VIP顾客），只需要修改Queue类的实现，而不需要改变整体的程序结构。  
  
此外，程序的设计还考虑了一些边界情况，比如当一个窗口的顾客处理完毕而另一个窗口还有顾客的情况。这种全面的考虑使得程序能够在各种情况下都能正确运行，提高了程序的可靠性。

|  |
| --- |
| 附件、main部分函数 |
| int main() {    cout << "请输入一行正整数，其中第一数字N（N<=1000）为顾客总数，后面跟着N位顾客的编号。" << endl;  int sum;  cin >> sum;  Queue<int> Q\_A(1000);  Queue<int> Q\_B(1000);  for (int i = 0; i < sum; i++) {  int id;  cin >> id;  if (Q\_A.CheckQuene(id)) {  Q\_A.EnQueue(id);  }  else {  Q\_B.EnQueue(id);  }  }  cout << "A窗口顾客为：" << endl;  Q\_A.Print();  cout << "B窗口顾客为：" << endl;  Q\_B.Print();  cout << "按下enter开始办理业务" << endl;  wait\_for\_enter();  cout << "办理结束，办理结束先后顺序为：" << endl;  //业务处理模拟部分  Q\_out.Print();  #ifdef \_WIN32  system("pause");  #elif defined(\_\_linux\_\_) || defined(\_\_APPLE\_\_)  cout << "按回车键继续..." << endl;  cin.get();  #endif  return 0;  } |

这个项目的实现不仅满足了题目的所有要求，还展示了良好的编程实践，包括模块化设计、错误处理、以及对各种情况的全面考虑。这种实现方式不仅解决了当前的问题，还为未来可能的扩展和修改提供了良好的基础。

# 项目总结

在本次数据结构课程设计中，我成功地设计并实现了一个功能完备的银行业务处理模拟系统。通过这个项目，我不仅巩固了对数据结构和算法的理解，还提升了面向对象编程和系统设计的能力。以下是对整个项目的详细总结：

## 8.1 学习体会

通过将课堂上学到的数据结构和算法知识应用到实际项目中，我深刻体会到了理论知识在解决实际问题中的重要性。在实践中我发现，理论知识为我们提供了解决问题的工具和方法，而实际应用则检验了我们对这些知识的理解和掌握程度。

项目让我更加熟悉了面向对象的设计原则，如封装、继承和多态。通过定义清晰的类和对象，我提高了代码的可读性和可维护性。面向对象的方法使得代码结构更加清晰，易于管理和扩展。

在项目开发过程中，我学习了如何规划和设计一个完整的系统，包括需求分析、系统架构设计、模块划分等。这些经验对于我未来成为一名软件工程师具有重要的意义。

## 8.2 技术收获

通过本项目，我提高了C++编程技能，特别是在STL的使用、异常处理和输入输出操作等方面。这些技能的提升，让我能够更加高效地编写高质量的代码。

我深入理解了链表、队列等数据结构的特点和适用场景，掌握了它们在实际问题中的应用方法。特别是在处理动态数据时，这些数据结构的灵活性和高效性显得尤为重要。

项目中实现了多种算法，如排序算法、查找算法等，提高了我的算法设计和优化能力。这些算法在处理大量数据时，能够有效提升程序的性能。

## 8.3 问题与挑战

在处理大量数据时，如何优化算法和数据结构以提高系统性能是一个挑战。我通过优化链表操作和算法逻辑来解决这一问题。例如，在处理顾客信息时，我采用了高效的排序算法，确保了数据的快速处理。

在用户输入错误或系统异常情况下，如何保证系统的健壮性是一个难题。我通过增加输入验证和异常捕获机制来提高系统的稳定性。这些措施确保了系统在面对错误输入或异常情况时，能够给出正确的反馈并保持稳定运行。

设计一个用户友好的界面也是项目中的一个挑战。我通过不断迭代和改进用户界面，使其更加直观和易用。用户界面的设计让我学会了如何从用户的角度思考问题，提升了用户体验。

## 8.4 改进方向

未来可以增加更多功能，如数据导出、数据导入、成绩查询等，使其更符合实际应用需求。这些功能的增加将使系统更加完善，满足更多用户的需求。

可以探索更高效的数据结构和算法，进一步提高系统的处理速度和响应能力。例如，可以考虑使用更高效的排序算法或者并行处理技术来提升性能。

可以进一步优化用户界面和交互设计，提供更加友好和直观的用户体验。这包括优化界面布局、提供更多的用户指引和帮助信息等。

## 8.5 结论

这次课程设计是一次宝贵的学习经历。它不仅让我将所学知识应用于实践，还让我学会了如何分析和解决实际问题。我克服了各种挑战，最终完成了一个功能完善、性能良好的考试报名系统。我相信，这次经历将对我未来的学习和工作产生积极的影响。