1. 读取职业

```cpp
void read_occupation(std::string file_path,std::map<std::string, std::string>& keyValueMap){
    std::ifstream file(file_path); // 打开名为"occupation.txt"的文件
    if (!file.is_open()) {
        std::cerr << "cant not open the file." << std::endl;
        return;
    }
    std::string line;
    while (std::getline(file, line)) {
        size_t pos = line.find(':');
        if (pos != std::string::npos) {
            std::string key = line.substr(0, pos);
            std::string value = line.substr(pos + 1);
            keyValueMap[key] = value;
        }
    }

    file.close(); // 关闭文件
}
```

2. user

```cpp

class UserInfo {
public:
    enum class AgeRange {
        UNDER_18,
        AGE_18_24,
        AGE_25_34,
        AGE_35_44,
        AGE_45_49,
        AGE_50_55,
        ABOVE_56
    };

    UserInfo(int userID, char gender, AgeRange age, int occupation, int zipCode)
            : userID(userID), gender(gender), age(age), occupation(occupation), zipCode(zipCode) {}

    [[nodiscard]] int getUserID() const { return userID; }
    [[nodiscard]] char getGender() const { return gender; }
    [[nodiscard]] AgeRange getAge() const { return age; }
    [[nodiscard]] int getOccupation() const { return occupation; }
    [[nodiscard]] int getZipCode() const { return zipCode; }

private:
    int userID;
    char gender;
    AgeRange age;
    int occupation;
    int zipCode;
};

```

```cpp
class BitCompressor {
public:
    BitCompressor() : data_(0), numBits_(0) {}

    void Append(uint32_t value, uint8_t numBits) {
        if (numBits_ + numBits > 32) {
            // Not enough space in the current 32 bits, flush and append to the next 32 bits.
            Flush();
        }
        data_ |= (value & ((1U << numBits) - 1)) << numBits_;
        numBits_ += numBits;
    }

    void Flush() {
        if (numBits_ > 0) {
            compressedData_.push_back(data_);
            data_ = 0;
            numBits_ = 0;
        }
    }

    [[nodiscard]] const std::vector<uint32_t>& GetCompressedData() const {
        return compressedData_;
    }

private:
    uint32_t data_;
    uint8_t numBits_;
    std::vector<uint32_t> compressedData_;
};

```

```cpp
int main() {
    MyError error;
    std::map<std::string, std::string> occupation;
    std::vector<UserInfo> users;
    read_occupation(DATADIR+"occupation.txt",occupation,error);
    if(!error.pass()){
        std::cout<<error.getErrorCode()<<" "<<error.getErrorDescription()<<std::endl;
    }
    read_user_info(DATADIR+"users.dat",users,error);
    if(!error.pass()){
        std::cout<<error.getErrorCode()<<" "<<error.getErrorDescription()<<std::endl;
    }


    BitCompressor compressor;

    // 压缩数据
    for (const UserInfo& user : users) {
        compressor.Append(user.getUserID(), 16);
        compressor.Append(user.getGender() == 'M' ? 0 : 1, 1);
        compressor.Append(static_cast<uint32_t>(user.getAge()), 3);
        compressor.Append(user.getOccupation(), 10);
        compressor.Append(user.getZipCode(), 32);
    }

    // 结束并刷新压缩器
    compressor.Flush();

    // 获取压缩后的数据
    const std::vector<uint32_t>& compressedData = compressor.GetCompressedData();

    // 输出压缩前的数据大小
    std::cout << "before zip: " << users.size() * sizeof(UserInfo) << " bytes" << std::endl;

    // 输出压缩后的数据大小
    std::cout << "after zip: " << compressedData.size() * sizeof(uint32_t) << " bytes" << std::endl;
    return 0;
}
```

```
before zip: 120800 bytes
after zip: 48320 bytes
```

```
Compressed Value: 00000000101000010000000000000001
Compressed Value: 00000000000000000000000000000000
Compressed Value: 00000001000011000000000000000010
Compressed Value: 00000000000000000000000000000000
Compressed Value: 00000000111101000000000000000011
Compressed Value: 00000000000000000000000000000000
Compressed Value: 00000000011110000000000000000100
Compressed Value: 00000000000000000000000000000000
Compressed Value: 00000001010001000000000000000101
Compressed Value: 00000000000000000000000000000000
Compressed Value: 00000000100110110000000000000110
Compressed Value: 00000000000000000000000000000000
Compressed Value: 00000000000101100000000000000111
Compressed Value: 00000000000000000000000000000000
Compressed Value: 00000000110001000000000000001000
Compressed Value: 00000000000000000000000000000000
Compressed Value: 00000001000101000000000000001001
```

```cpp


#ifndef DATA_DEAL_COMPRESSOR_H
#define DATA_DEAL_COMPRESSOR_H


```

```cpp
#include <iostream>
#include <vector>
#include <cstdint>
#include <bitset>
class BitCompressor {
public:
    BitCompressor() : data_(0), numBits_(0) {}

    template <typename T>
    void Append(T value, uint8_t numBits) {
        if (numBits_ + numBits > 32) {
            // 如果当前 32 位不够存放，需要刷新
            Flush();
        }
        data_ |= (value & ((1U << numBits) - 1)) << numBits_;
        numBits_ += numBits;
    }

    void Flush() {

        if (numBits_ > 0) {
            // 将当前 32 位添加到压缩数据
            compressedData_.push_back(data_);
            data_ = 0;
            numBits_ = 0;
        }
    }

    [[nodiscard]] const std::vector<uint32_t>& GetCompressedData() const {
        return compressedData_;
    }

private:
    uint32_t data_;               // 用于存放当前 32 位数据
    uint8_t numBits_;             // 当前已存的比特数
    std::vector<uint32_t> compressedData_;  // 存放压缩后的数据
};

class BitDecompressor {
public:
    explicit BitDecompressor(const std::vector<uint32_t>& compressedData)
            : compressedData_(compressedData), dataIndex_(0), currentData_(0), currentBit_(0) {}

    template <typename T>
    bool GetNextValue(T& value, uint8_t numBits) {
        if (dataIndex_ >= compressedData_.size() * 32) {
            return false; // 已经处理完所有数据
        }

        // 从当前数据中提取 numBits 位数据
        uint32_t mask = (1U << numBits) - 1;
        value = static_cast<T>((compressedData_[dataIndex_ / 32] >> currentBit_) & mask);

        // 更新当前位偏移和索引
        currentBit_ += numBits;
        if (currentBit_ >= 32) {
                currentBit_ = 0;
                dataIndex_++;
            }

            return true;
        }

private:
    const std::vector<uint32_t>& compressedData_;
    size_t dataIndex_;
    uint32_t currentData_;
    uint8_t currentBit_;
};

#endif //DATA_DEAL_COMPRESSOR_H

```