班级：2022211301　　**姓名：**卢安来　　**学号：**2022212720

1、根据课堂教学内容，更正 fifo_queue.c。

问题一：命名问题

代码到处充斥着 fifo_queue 这一字符串，但实际上 queue 指队列，本身就是先进先出（FIFO）的数据结构，非先进先出的队列如双端队列 deque 和优先队列 priority_queue 均有其他的英文拼写，在 queue 前加上 fifo_前缀实属**无意义重复**，应当去除。

代码第 5 行到第 9 行定义了结构类型 fifo_queue_item，其成员 item_value 和 next_item 的命名含有 item_和_item，实际与类型名 fifo_queue_item 造成了上下文的**无意义重复**，应当去除。

```
struct fifo_queue_item
{
   int item_value;
   struct fifo_queue_item* next_item;
};
```

改正后为

```
/**
 * @brief Structure representing an item in the queue.
 */
struct queue_item {
    int value;               /**< Value of the queue item */
    struct queue_item *next; /**< Pointer to the next item in the queue */
};
```

代码第 11 行到第 15 行定义了结构类型 fifo_queue，其成员 queue_head 和 queue_tail 的命名含有 queue_，实际与类型名 fifo_queue 造成了上下文的**无意义重复**，应当去除。

```
struct fifo_queue
{
   struct fifo_queue_item* queue_head;
   struct fifo_queue_item* queue_tail;
};
```

改正后为

```
/**
 * @brief Structure representing the queue.
 */
struct queue {
    struct queue_item *head; /**< Pointer to the head of the queue */
    struct queue_item *tail; /**< Pointer to the tail of the queue */
};
```

代码第 18 行和第 33 行声明（并定义）了函数 `fifo_queue_push_back` 和 `fifo_queue_pop_front`，用于向队列中插入元素和从队首弹出元素，但根据所学知识，队列仅能从队尾插入元素，也仅能从队首弹出元素，`_back` 和`_front` 属于**无意义重复**，应当去除。

```
// push a item_value to queue back
void fifo_queue_push_back(struct fifo_queue* queue, int item_value)

// pop a item_value from queue front
int fifo_queue_pop_front(struct fifo_queue* queue, int* value)
```

改正后为

```
/**
 * @brief Pushes a value onto the back of the queue.
 *
 * @param q Pointer to the queue.
 * @param value Value to be pushed onto the queue.
 * @return int 1 if the operation was successful, 0 otherwise.
 */
int queue_push(struct queue *const q, const int value)

/**
 * @brief Pops a value from the front of the queue.
 *
 * @param q Pointer to the queue.
 * @param result Pointer to an integer where the popped value will be stored.
 * @return int 1 if the operation was successful, 0 otherwise.
 */
int queue_pop(struct queue *const q, int *const result)
```

问题二：函数处理逻辑问题

代码第 35 行到第 50 行存在复杂 `if-else` 嵌套，应当提前 `return` 降低代码嵌套层次，提高可读性。

```
int fifo_queue_pop_front(struct fifo_queue* queue, int* value)
{
   if(queue->queue_head != NULL)
   {
      struct fifo_queue_item* item = queue->queue_head;
      if(item->next_item == NULL)
      {
         queue->queue_head = NULL;
         queue->queue_tail = NULL;
      }
      else
         queue->queue_head = item->next_item;
      *value = item->item_value;
      free(item);
      return 1;
   }
   else
      return 0;
}
```

改正后为

```
int queue_pop(struct queue *const q, int *const result) {
    if (q->head == NULL)
        return 0;
    struct queue_item *item = q->head;
    if (item->next == NULL)
        q->head = q->tail = NULL;
    else
        q->head = item->next;

    *result = item->value;
    free(item);
    return 1;
}
```

代码第 23 行到第 29 行的提前 `return` 可读性不佳，应当将 `return` 跳过部分作为 `else` 分支。

```
   if(queue->queue_head == NULL)
   {
      queue->queue_head = queue->queue_tail = item;
      return;
   }
   queue->queue_tail->next_item = item;
   queue->queue_tail = item;
```

改正后为

```
    if (q->head == NULL) {
        q->head = q->tail = item;
    } else {
        q->tail->next = item;
        q->tail = item;
    }
```

代码未对第 20 行 `malloc` 函数返回的指针进行检查，应当增添检查部分，修改函数实现。

```
   struct fifo_queue_item* item = (struct fifo_queue_item*)malloc(sizeof(struct
fifo_queue_item));
   item->item_value = item_value;
   item->next_item = NULL;
```

改正后为

```
    struct queue_item *item = malloc(sizeof(struct queue_item));
    if (item == NULL)
        return 0;

    *item = (struct queue_item){.value = value, .next = NULL};
```

问题三：Magic Number 问题

代码第 59 行和第 64 行 for 循环的循环条件中存在 magic number（10 和 11），应当更正。

```c
int main()
{
   struct fifo_queue queue;
   queue.queue_head = queue.queue_tail = NULL;

   int i;
   for(i = 0; i < 10; i++)
   {
      fifo_queue_push_back(&queue, i);
      printf("push to back: %d\n", i);
   }
   for(i = 0; i < 11; i++)
   {
      int value;
      int success = fifo_queue_pop_front(&queue, &value);
      if(success)
         printf("pop from front: %d\n", value);
      else
         printf("pop from front fail\n");
   }
}
```

改正后为

```c
#define TEST_TIMES 10

/**
 * @brief Main function to test the queue operations.
 *
 * @return int Exit status of the program.
 */
int main() {
    struct queue q = {.head = NULL, .tail = NULL};

    for (int i = 0; i < TEST_TIMES; i++)
        if (queue_push(&q, i))
            printf("push to back: %d\n", i);
        else
            printf("push to back fail\n");

    for (int i = 0; i < TEST_TIMES + 1; i++) {
        int value;
        if (queue_pop(&q, &value))
            printf("pop from front: %d\n", value);
        else
            printf("pop from front fail\n");
    }

    return 0;
}
```

建议一：代码第 18 行和第 33 行声明（并定义）了函数 `fifo_queue_push_back` 和 `fifo_queue_pop_front`，其形参类型缺少底层或顶层 `const` 限定，容易造成意料之外的修改，应当补充。

建议二：代码第 18 行和第 33 行声明（并定义）了函数 `fifo_queue_push_back` 和 `fifo_queue_pop_front`，其形参名与类型名相同，容易造成歧义，应当替换。

建议三：代码第 20 行使用强制类型转换对 `malloc` 函数返回的指针进行了转换，但 C 语言中 `void*`可隐式类型转换到任意类型指针，添加强制类型转换属于冗长的无意义操作，应当去除。

建议四：代码第 59 行和第 64 行 `for` 循环的循环变量定义在外部，存在作用域扩大风险，应当修改定义位置，调整至 `for` 循环初始化语句处。

建议五：代码第 67 行的 `success` 变量冗余，应当去除，可直接弹出函数的返回结果作为 `if` 语句的条件。

建议六：代码第 21, 22, 40, 41 和 56 行的赋值语句频繁出现变量名的无意义冗余，应当使用复合字面量改写。

建议七：代码提供的数据结构的定义和接口均无良好注释，应补充以便于调用时的编辑器提示。

建议八：代码使用三字符宽度空格缩进，不太合适，应当替换。

修改后的程序见下图或上传文件。修改完成的代码可以通过 cpplint 的审查。

```c
#include <stdio.h>
#include <stdlib.h>

/**
 * @brief Structure representing an item in the queue.
 */
struct queue_item {
    int value;              /**< Value of the queue item */
    struct queue_item *next; /**< Pointer to the next item in the queue */
};

/**
 * @brief Structure representing the queue.
 */
struct queue {
    struct queue_item *head; /**< Pointer to the head of the queue */
    struct queue_item *tail; /**< Pointer to the tail of the queue */
};

/**
 * @brief Pushes a value onto the back of the queue.
 *
 * @param q Pointer to the queue.
 * @param value Value to be pushed onto the queue.
 * @return int 1 if the operation was successful, 0 otherwise.
 */
int queue_push(struct queue *const q, const int value) {
    struct queue_item *item = malloc(sizeof(struct queue_item));
    if (item == NULL)
        return 0;

    *item = (struct queue_item){.value = value, .next = NULL};

    if (q->head == NULL) {
        q->head = q->tail = item;
    } else {
        q->tail->next = item;
        q->tail = item;
    }
    return 1;
}

/**
 * @brief Pops a value from the front of the queue.
 *
 * @param q Pointer to the queue.
 * @param result Pointer to an integer where the popped value will be stored.
 * @return int 1 if the operation was successful, 0 otherwise.
 */
int queue_pop(struct queue *const q, int *const result) {
    if (q->head == NULL)
        return 0;
    struct queue_item *item = q->head;
    if (item->next == NULL)
        q->head = q->tail = NULL;
    else
        q->head = item->next;

    *result = item->value;
    free(item);
    return 1;
}

#define TEST_TIMES 10

/**
 * @brief Main function to test the queue operations.
 *
 * @return int Exit status of the program.
 */
int main() {
    struct queue q = {.head = NULL, .tail = NULL};

    for (int i = 0; i < TEST_TIMES; i++)
        if (queue_push(&q, i))
            printf("push to back: %d\n", i);
        else
            printf("push to back fail\n");

    for (int i = 0; i < TEST_TIMES + 1; i++) {
        int value;
        if (queue_pop(&q, &value))
            printf("pop from front: %d\n", value);
        else
            printf("pop from front fail\n");
    }

    return 0;
}

```

2、根据课堂所学内容，更正 serialize_json.cpp。

问题一：命名问题

代码中 escape_json 函数和 serialize 函数内的 std::ostringstream 变量命名过于简单，应当更正。

```
ostringstream o;
```

改正后为

```
std::ostringstream oss;
```

代码中 escape_json 函数和 serialize 函数的命名存在歧义或不明确，应该更名，例如改为 escape_to_json 和 serialize_to_json。

```
// 将字符串根据 json 的转义要求进行转义，返回一个新的字符串
string escape_json(const string &s)

// 将 map 中的数据序列化为一个 json 格式的字符串返回
string serialize(map<string, string>& m)
```

改正后为

```
/**
 * @brief Escapes JSON special characters in a string.
 *
 * This function takes a string and returns a new string with all the JSON special characters
 * properly escaped according to the JSON specification.
 *
 * @param s The input string to escape.
 * @return std::string The escaped string.
 */
std::string escape_to_json(const std::string &s)

/**
 * @brief Serializes a map of strings to a JSON string.
 *
 * This function takes a map of strings and serializes it into a JSON formatted string.
 *
 * @param m The map to serialize.
 * @return std::string The JSON formatted string.
 */
std::string serialize_to_json(const std::map<std::string, std::string> &m)
```

问题二：表达式不够自然

代码第 22 行条件表达式可读性差且逻辑有误，应当仅直接输出可打印符，而非直接输出所有非 ASCII 字符，应用 std::isprint 更正。

```
            if (!('\x00' <= *c && *c <= '\x1f'))
                o << *c;
```

改正后为

```
            if (std::isprint(c))
                oss << c;  // If the character is printable, add it as is
```

代码中 escape_json 函数和 serialize 函数内的 std::ostringstream 流输出语句过长，可读性较低，应该适度换行。

```
                o << "\\u" << hex << setw(4) << setfill('0') << (int)*c;
// some code
     o << "\"" << escape_json(i->first) << "\":\"" << escape_json(i->second) << "\"";
```

改正后为

```
                oss << "\\u"  // Otherwise, output as a Unicode escape sequence
                    << std::hex
                    << std::setw(4)
                    << std::setfill('0')
                    << static_cast<unsigned>(c);
// some code
        oss << "\"" << escape_to_json(key) << "\""
            << ":"
            << "\"" << escape_to_json(value) << "\"";
```

问题三：否定之否定

第 37 行的变量 not_first 的逻辑存在「否定之否定」，不符合思维习惯，应当简化。

```
   bool not_first = false;
   for(i = m.begin(); i != m.end(); i++){
      if(not_first)
         o << ",";
      else
         not_first = true;
```

改正后为

```
    bool is_first = true;
    for (const auto &[key, value] : m) {
        if (is_first)
            is_first = false;
        else
            oss << ",";
```

问题四：循环遍历的问题

代码中 escape_json 函数和 serialize 函数内对 STL 容器的遍历方式较为繁琐，应当更正。

```
for(const char* c = s.c_str(); *c != 0; c++)

for(i = m.begin(); i != m.end(); i++)
```

改正后为

```
for (const char c : s)

for (const auto &[key, value] : m)
```

建议一：使用了 `using namespace std;`，可能造成命名空间污染，导致大量潜在的作用域问题，应当更正。

建议二：代码第 25 行进行了 C 语言的强制类型转换，C++代码应该使用 `static_cast<type>(value)`进行强制类型转换。且此处类型应当限定为无符号型，否则逻辑有误，应当更正。

建议三：`serialize` 函数的前面应该给引用加上 `const` 限定，以防修改。

建议四：`serialize` 函数的循环中存在 `i->first` 和 `i->second` 这种可读性不太好的用法，应当使用结构化绑定增强代码可读性。

建议五：`main` 函数内测试数据的初始化存在变量名的多次无意义重复，应当简化。

建议六：代码提供的数据结构的定义和接口均无良好注释，应补充以便于调用时的编辑器提示。

建议七：代码使用三字符宽度空格缩进，不太合适，应当替换。

修改后的程序见下图或上传文件。修改完成的代码可以通过 cpplint 的审查。

```cpp
#include <cctype>
#include <iomanip>
#include <iostream>
#include <map>
#include <sstream>

/**
 * @brief Escapes JSON special characters in a string.
 *
 * This function takes a string and returns a new string with all the JSON special characters
 * properly escaped according to the JSON specification.
 *
 * @param s The input string to escape.
 * @return std::string The escaped string.
 */
std::string escape_to_json(const std::string &s) {
    std::ostringstream oss;
    for (const char c : s) {
        switch (c) {
        case '"':
            oss << "\\\"";
            break;
        case '\\':
            oss << "\\\\";
            break;
        case '\b':
            oss << "\\b";
            break;
        case '\f':
            oss << "\\f";
            break;
        case '\n':
            oss << "\\n";
            break;
        case '\r':
            oss << "\\r";
            break;
        case '\t':
            oss << "\\t";
            break;
        default:
            if (std::isprint(c))
                oss << c;  // If the character is printable, add it as is
            else
                oss << "\\u"  // Otherwise, output as a Unicode escape sequence
                    << std::hex
                    << std::setw(4)
                    << std::setfill('0')
                    << static_cast<unsigned>(c);
        }
    }
    return oss.str();
}

/**
 * @brief Serializes a map of strings to a JSON string.
 *
 * This function takes a map of strings and serializes it into a JSON formatted string.
 *
 * @param m The map to serialize.
 * @return std::string The JSON formatted string.
 */
std::string serialize_to_json(const std::map<std::string, std::string> &m) {
    std::ostringstream oss;
    oss << "{";
    bool is_first = true;
    for (const auto &[key, value] : m) {
        if (is_first)
            is_first = false;
        else
            oss << ",";

        oss << "\"" << escape_to_json(key) << "\""
            << ":"
            << "\"" << escape_to_json(value) << "\"";
    }
    oss << "}";
    return oss.str();
}

/**
 * @brief Main entry point of the program.
 *
 * This is the main function that demonstrates the serialization of a map to a JSON string.
 *
 * @return int The exit code of the program.
 */
int main() {
    std::map<std::string, std::string> m = {
        {"sjdf", "sk\ndjfksdf"},
        {"hsdhf", "sjdgfjsdhgfj sdgfjsd"},
        {"sjdh", "sjdhfg\x01sjdf sdf"},
        {"uery", "uerytjfsd"},
        {"jsdkf", "123456"},
        {"sdhs", "ksjhdfksf"},
    };

    std::string json = serialize_to_json(m);
    std::cout << json << std::endl;
    return 0;
}

```

3、根据课堂所学内容，更正 split_str.cpp。

问题一：命名问题

process_str 的命名不够清晰，没有明确地表达该函数实现的功能，应该更名为 split_str。

```
// 将一个长字符串，按照空白字符分开，存放在一个字符串数组中
// 返回分解出来的短字符串数量
// 输入参数 argv，存放返回的字符串的数组
// 输入参数 buffer，输入的长字符串
// 返回值：正确返回>0，错误返回 0
int process_str(char* argv[], const char* buffer)
```

改正后为

```
/**
 * @brief Splits a given string into an array of arguments.
 *
 * This function parses the input string and separates it into individual arguments
 * based on whitespace. It stores the arguments in the provided argv array.
 *
 * @param argv Array to store the pointers to the arguments.
 * @param buffer The string to be split into arguments.
 * @return The number of arguments parsed, or 0 if an error occurs.
 */
int split_str(char *const argv[], const char *buffer)
```

问题二：Magic Number 问题

代码中存在 30、31、32 等 magic number，应当更正。

添加宏定义如下（其余部分修正见最终代码）

```
/// Maximum number of arguments that can be parsed.
#define MAX_ARGC 30
/// Maximum length of each argument (including \0).
#define MAX_ARG_LEN 32
```

问题三：处理逻辑问题

代码中对空白字符的判定有逻辑问题，仅判定了空格和制表符，应当使用 std::isblank 进行非终结符空白字符判定。

```
while(buffer[i] != ' ' && buffer[i] != '\t' && buffer[i] != 0)
```

改正后为

```
while (std::isblank(*p))
```

buffer[i]在代码中的大量重复，影响代码可读性，应当更换

为指针。代码中存在对 `j>=31` 的多重 `break`，循环嵌套层数过深，逻辑复杂，应当简化。

```
int process_str(char* argv[], const char* buffer)
{
   int argc = 0;
   int j = 0;
   int i = 0;

   while(buffer[i] != 0)
   {
      while(buffer[i] == ' ' || buffer[i] == '\t' )
         i++;
      if(buffer[i] != 0)
      {
         while(buffer[i] != ' ' && buffer[i] != '\t' && buffer[i] != 0)
         {
            argv[argc][j] = buffer[i];
            i++;
            j++;
            if(j >= 31)
               break;
         }
         if(j >= 31)
            break;
         argv[argc][j] = 0;
         argc++;
         j = 0;
         if(argc >= 30)
            break;
      }
   }
   if(buffer[i] != 0)
   {
      printf("invalid command.\n");
      return 0;
   }
   return argc;
}
```

改正后为

```
int split_str(char *const argv[], const char *buffer) {
    int argc = 0;
    const char *p = buffer;
    while (*p) {
        while (std::isblank(*p))
            ++p;

        if (!(*p))
            break;

        if (argc == MAX_ARGC - 1) {
            std::fprintf(stderr, "[ERROR] more than %d arguments.\n", MAX_ARGC);
            return 0;
        }

        int i = 0;
        while (*p && !std::isblank(*p)) {
            argv[argc][i++] = *p++;
            if (i + 1 >= MAX_ARG_LEN) {
                std::fprintf(stderr,
                             "[ERROR] argument longer than %d.\n", MAX_ARG_LEN);
                return 0;
            }
        }
        argv[argc][i] = '\0';
        ++argc;
    }

    return argc;
}
```

内存管理存在问题，申请的内存未释放，应当更正。

```
int main()
{
   char* argv[30];
   int i;
   for(i = 0; i < 30; i++)
      argv[i] = new char[32];
   int argc = process_str(argv, " kjsf  ks  ks dhf ksdjh ksdjfh skdjf skdf skdjf sdkjf
kjsdhf   ");
   for(i = 0; i < argc; i++)
      printf("argv[%d] = %s\n", i, argv[i]);
}
```

改正后为

```
/**
 * @brief Main function demonstrating the usage of split_str.
 *
 * This function initializes an array of char pointers, calls split_str to parse a string,
 * and then prints out the parsed arguments. It also cleans up the allocated memory.
 *
 * @return Always returns 0.
 */
int main() {
    char *argv[MAX_ARGC];
    for (int i = 0; i < MAX_ARGC; i++)
        argv[i] = new char[MAX_ARG_LEN];

    int argc = split_str(
        argv,
        " kjsf  ks  ks dhf ksdjh ksdjfh skdjf skdf skdjf sdkjf kjsdhf   ");

    for (int i = 0; i < argc; i++)
        std::printf("argv[%d] = %s\n", i, argv[i]);

    for (int i = 0; i < MAX_ARGC; i++) {
        delete[] argv[i];
        argv[i] = nullptr;
    }

    return 0;
}
```

建议一：代码第 1 行，C++源程序不应使用 C 语言头文件，而应该使用 C++提供的 C 库设置，`stdio.h` 应改为 `cstdio`。

建议二：代码提供的注释可以再优化，应该使用可以生成文档和编辑器提示的注释。

建议三：`process_str` 的形参类型未添加 `const` 限定。

建议四：代码使用三字符宽度空格缩进，不太合适，应当替换。

建议五：代码第 49 行和第 52 行 `for` 循环的循环变量定义在外部，存在作用域扩大风险，应当修改定义位置，调整至 `for` 循环初始化语句处。

建议六：判定空字符用`!=0` 略显累赘，应当简化。

建议七：赋值空字符用 `0` 而非`'\0'`，表意不明确，应当更正。

建议八：报错信息导入到了标准输出流 `stdout` 而非标准错误流 `stderr`，应当更正。

修改后的程序见下图或上传文件。修改完成的代码可以通过 cpplint 的审查。

```cpp
#include <cctype>
#include <cstdio>

/// Maximum number of arguments that can be parsed.
#define MAX_ARGC 30
/// Maximum length of each argument (including \0).
#define MAX_ARG_LEN 32

/**
 * @brief Splits a given string into an array of arguments.
 *
 * This function parses the input string and separates it into individual arguments
 * based on whitespace. It stores the arguments in the provided argv array.
 *
 * @param argv Array to store the pointers to the arguments.
 * @param buffer The string to be split into arguments.
 * @return The number of arguments parsed, or 0 if an error occurs.
 */
int split_str(char *const argv[], const char *buffer) {
    int argc = 0;
    const char *p = buffer;
    while (*p) {
        while (std::isblank(*p))
            ++p;

        if (!(*p))
            break;

        if (argc == MAX_ARGC - 1) {
            std::fprintf(stderr, "[ERROR] more than %d arguments.\n", MAX_ARGC);
            return 0;
        }

        int i = 0;
        while (*p && !std::isblank(*p)) {
            argv[argc][i++] = *p++;
            if (i + 1 >= MAX_ARG_LEN) {
                std::fprintf(stderr,
                             "[ERROR] argument longer than %d.\n", MAX_ARG_LEN);
                return 0;
            }
        }
        argv[argc][i] = '\0';
        ++argc;
    }

    return argc;
}

/**
 * @brief Main function demonstrating the usage of split_str.
 *
 * This function initializes an array of char pointers, calls split_str to parse a string,
 * and then prints out the parsed arguments. It also cleans up the allocated memory.
 *
 * @return Always returns 0.
 */
int main() {
    char *argv[MAX_ARGC];
    for (int i = 0; i < MAX_ARGC; i++)
        argv[i] = new char[MAX_ARG_LEN];

    int argc = split_str(
        argv,
        " kjsf  ks  ks dhf ksdjh ksdjfh skdjf skdf skdjf sdkjf kjsdhf   ");

    for (int i = 0; i < argc; i++)
        std::printf("argv[%d] = %s\n", i, argv[i]);

    for (int i = 0; i < MAX_ARGC; i++) {
        delete[] argv[i];
        argv[i] = nullptr;
    }

    return 0;
}

```