

栈的应用-括号匹配实验报告

专业班级：计算机 212

学号：2109010215

姓名：杜思维

程序源码

main.c:

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

#include "LinkStack.h"
#include "main_loop.h"

/**
 * @brief 链栈指针
 *
 */
LinkStack *pStack;

/**
 * @brief 字符串长度
 *
 */
#define STR_SIZE 100

/**
 * @brief 输出分割符
 *
 * @param count
 */
void print_split(char count)
{
    for (char i = 0; i < count; i++)
        putchar('-');
    putchar('\n');
}

/**
 * @brief 输出错误提示
 *
 * @param index 索引
 * @param bracket 括号
 */
void print_error(index_t index, char bracket)
{
    if (index || !pStack)
    {
        printf("第 %d 个字符的括号 %c ", index + 1, bracket);
        if (pStack)
            printf("和");
    }
}
```

```
    if (pStack)
        printf("第 %d 个字符的括号 %c ",
               top(pStack).index + 1,
               top(pStack).bracket);
    puts("不匹配!");
}

/**
 * @brief 括号判定宏定义
 *
 * @param left 左括号
 * @param right 右括号
 */
#define BRACKET_CASE(left, right) \
    case right: \
        if (empty(pStack) || top(pStack).bracket != left) \
        { \
            print_error(i, str[i]); \
            return; \
        } \
        pStack = pop(pStack); \
        break;

/**
 * @brief 输入数据
 *
 * @param str 字符串指针
 */
void input_data(char *str)
{
    FILE *file = fopen("expr.txt", "r");

    bool is_read_file = true;
    if (file)
    {
        printf("检测到 expr.txt 文件, 按回车读入, 输入 0 跳过: ");
        fflush(stdin);
        is_read_file = getchar() - '0';
        fflush(stdin);
    }
    else
        is_read_file = false;

    if (is_read_file)
    {
        fgets(str, STR_SIZE, file);
        printf("读入的表达式为: %s\n", str);
    }
    else
    {
        printf("请输入表达式, 不支持中文, 至多 %d 个字符:\n", STR_SIZE);
    }
}
```

```
        fflush(stdin);
        gets(str);
    }
}

/**
 * @brief 主循环回调
 *
 */
MAIN_LOOP_CALLBACK(loop_callback)
{
    puts("\t\t 括号匹配\n");
    puts("程序会自动检测运行目录中的 expr.txt 文件");
    print_split(40);

    char str[STR_SIZE] = "";
    input_data(str);

    pStack = initStack();

    for (index_t i = 0; str[i] != '\n' && str[i] != 0; i++)
    {
        if (str[i] == '(' || str[i] == '[' || str[i] == '{')
            pStack = push(pStack, INIT_DATA(str[i], i));
        else
            switch (str[i])
            {
                BRACKET_CASE('(', ')')
                BRACKET_CASE('[', ']')
                BRACKET_CASE('{', '}')
            }
    }

    if (empty(pStack))
        puts("括号匹配!");
    else
        print_error(0, 0);
}

/**
 * @brief 主函数
 *
 */
int main()
{
    main_loop(loop_callback);

    return 0;
}
```

main_loop.c:

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

#include "main_loop.h"

/**
 * @brief 主循环函数
 *
 * @param callback 回调
 */
void main_loop(main_loop_callback callback)
{
    bool is_continue = true;

    while (is_continue)
    {
        system("cls");

        callback();

        printf("按回车继续, 输入 0 退出: ");

        fflush(stdin);
        is_continue = getchar() - '0';
        fflush(stdin);
    }
}
```

main_loop.h:

```
#ifndef _MAIN_LOOP_
#define _MAIN_LOOP_

#include <stdbool.h>

/**
 * @brief 主循环回调
 *
 */
typedef void (*main_loop_callback)();

/**
 * @brief 主循环回调宏定义
 *
 */
#define MAIN_LOOP_CALLBACK(function_name) void function_name()

/**
 * @brief 主循环函数
 *
 */
```

```
* @param callback 回调
*/
void main_loop(main_loop_callback callback);

#endif // _MAIN_LOOP_
```

link_stack.c:

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

#include "LinkStack.h"

/**
 * @brief 初始化栈
 *
 * @return LinkStack* 栈指针, 始终为 NULL
 */
LinkStack *initStack()
{
    return NULL;
}

/**
 * @brief 判空
 *
 * @param s 栈指针
 * @return true 为空
 * @return false 不为空
 */
bool empty(LinkStack *s)
{
    return s == NULL;
}

/**
 * @brief 入栈
 *
 * @param s 栈指针
 * @param x 数据
 */
LinkStack *push(LinkStack *s, DataType x)
{
    LinkStack *p = (LinkStack *)malloc(sizeof(LinkStack));
    p->data = x;
    p->next = s;
    return p;
}

/**
 * @brief 出栈
```

```
*
* @param s 栈指针
*/
LinkStack *pop(LinkStack *s)
{
    LinkStack *p = s;
    s = s->next;
    free(p);
    return s;
}

/**
* @brief 取栈顶数据
*
* @param s 栈指针
* @return DataType 数据
*/
DataType top(LinkStack *s)
{
    return s->data;
}
```

link_stack.h:

```
#ifndef _LINK_STACK_
#define _LINK_STACK_

#include <stdbool.h>

/**
* @brief 索引/类型
*
*/
typedef char index_t;

/**
* @brief 数据类型
*
*/
typedef struct ExprUnit
{
    char bracket; //括号
    index_t index; //索引
} DataType;

/**
* @brief 初始化数据
*
*/
#define INIT_DATA(bracket_p, index_p) \
    (DataType)
```

```
{
    .bracket = bracket_p,
    .index = index_p
}

/**
 * @brief 栈结构体
 *
 */
typedef struct Node
{
    DataType data;    //数据域
    struct Node *next; //指针域
} LinkStack;

/**
 * @brief 初始化数据
 *
 */
#define INIT_DATA(bracket_p, index_p) \
    (DataType) \
    { \
        .bracket = bracket_p, \
        .index = index_p \
    }

/**
 * @brief 栈结构体
 *
 */
typedef struct Node
{
    DataType data;
    struct Node *next;
} LinkStack;

/**
 * @brief 初始化栈
 *
 * @return LinkStack* 栈指针, 始终为 NULL
 */
LinkStack *initStack();

/**
 * @brief 判空
 *
 * @param s 栈指针
 * @return true 为空
 * @return false 不为空
 */
bool empty(LinkStack *s);
```

```
/**
 * @brief 入栈
 *
 * @param s 栈指针
 * @param x 数据
 */
LinkStack *push(LinkStack *s, DataType x);

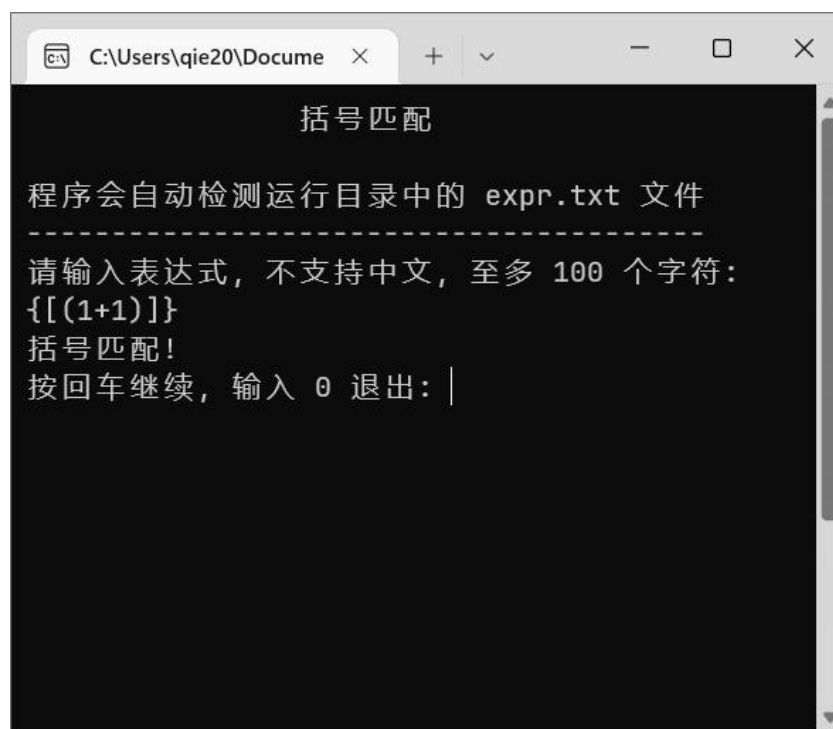
/**
 * @brief 出栈
 *
 * @param s 栈指针
 */
LinkStack *pop(LinkStack *s);

/**
 * @brief 取栈顶数据
 *
 * @param s 栈指针
 * @return DataType 数据
 */
DataType top(LinkStack *s);

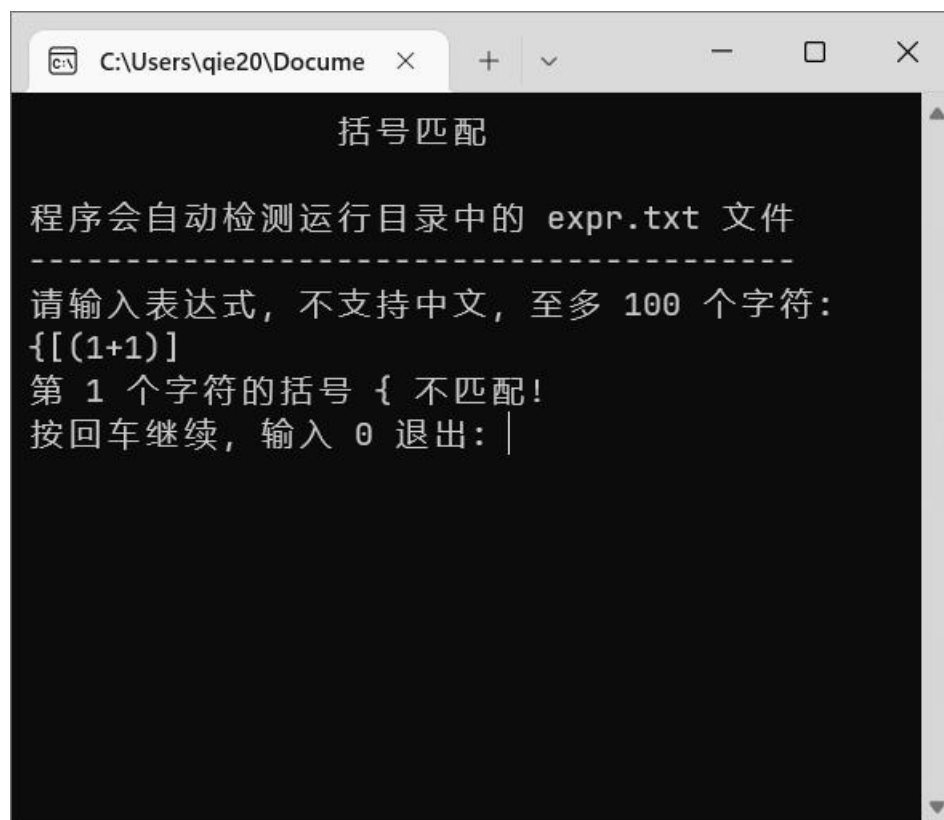
#endif // _LINK_STACK_
```

2. 运行结果截图

1. 括号匹配



2. 括号多余



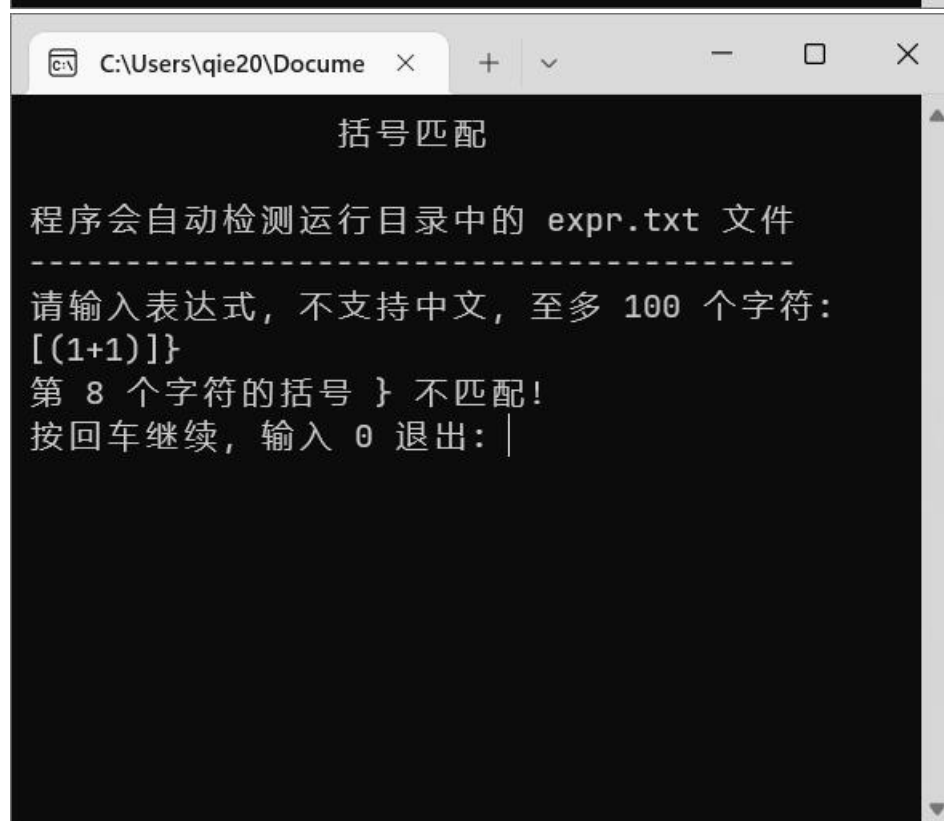
```

C:\Users\qie20\Docume  X  +  v  -  □  X

          括号匹配

程序会自动检测运行目录中的 expr.txt 文件
-----
请输入表达式, 不支持中文, 至多 100 个字符:
[(1+1)]
第 1 个字符的括号 { 不匹配!
按回车继续, 输入 0 退出: |

```



```

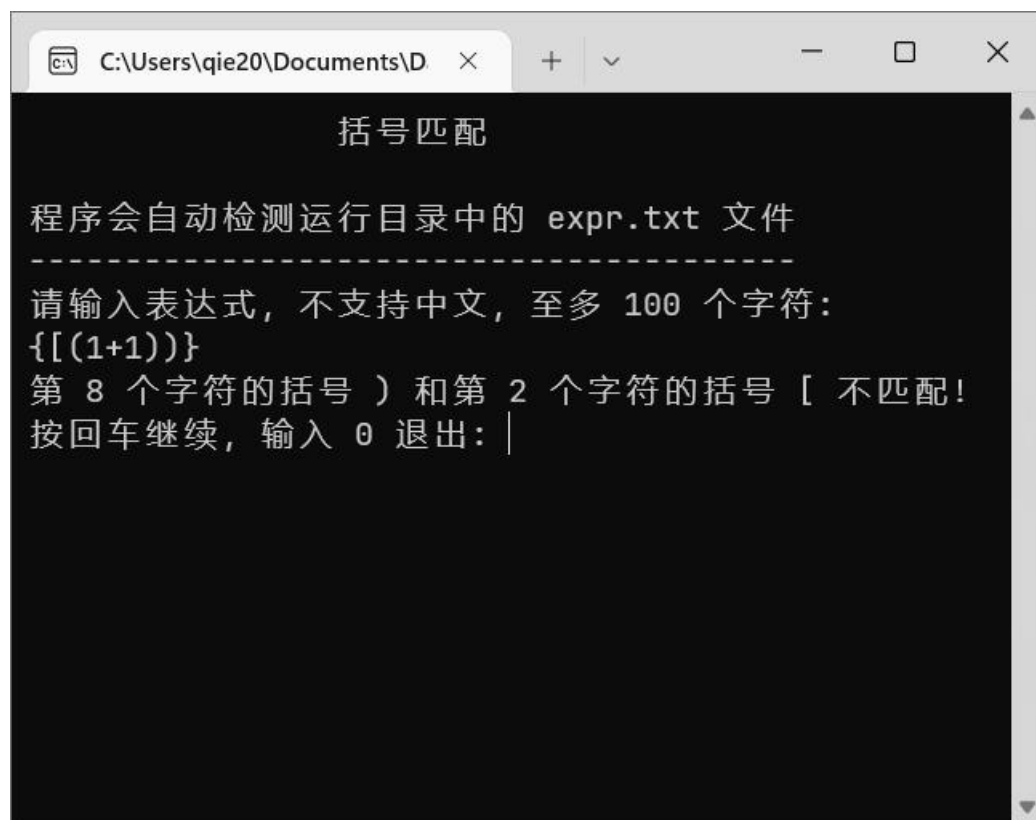
C:\Users\qie20\Docume  X  +  v  -  □  X

          括号匹配

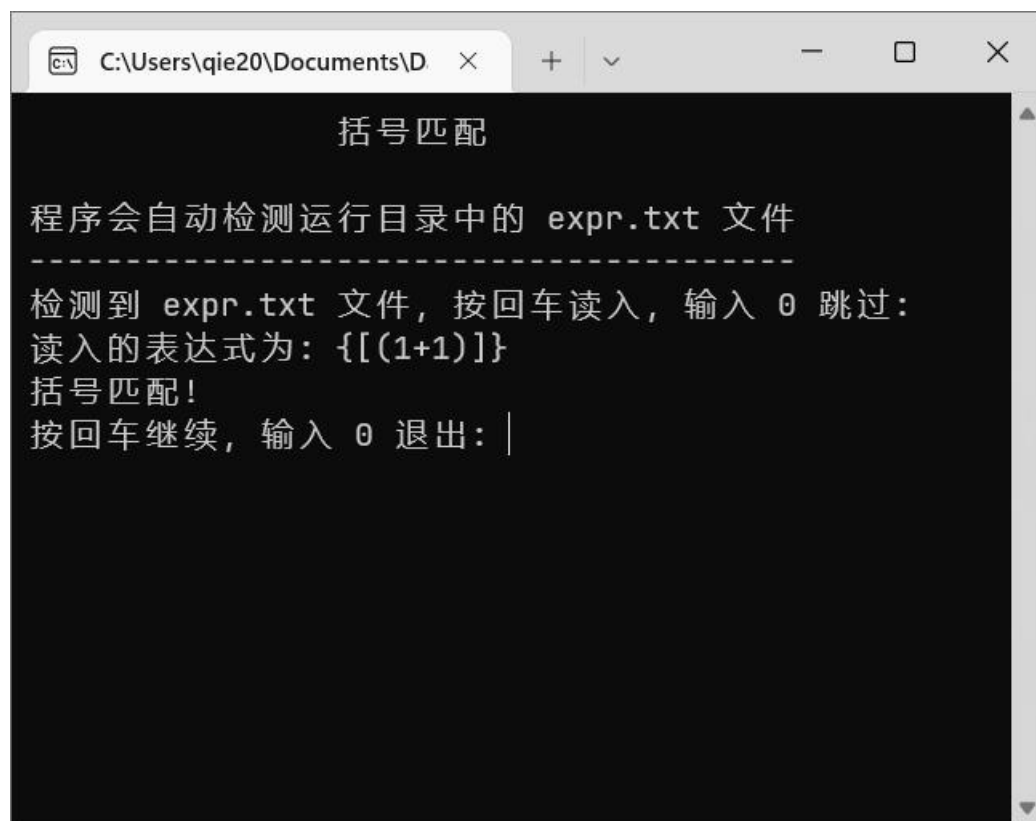
程序会自动检测运行目录中的 expr.txt 文件
-----
请输入表达式, 不支持中文, 至多 100 个字符:
[(1+1)]}
第 8 个字符的括号 } 不匹配!
按回车继续, 输入 0 退出: |

```

3. 括号不配对



4. 文件读入



3. 总结

- (1) 实验完成功能
 - (2) 实验创新点（例如更多种类的括号匹配等）
 - (3) 列举程序编写中遇到的问题，及解决方法
-
- (1) 1. 定义栈的数据类型；
2. 定义栈的初始化算法；
3. 定义栈的判空算法；
4. 定义出栈算法；
5. 定义入栈算法；
6. 定义取栈顶元素算法；
7. 调用栈的基本运算实现括号匹配的算法。
8. 使用更多种类的括号进行匹配检验。
9. 将检测的字符串或者字符数组保存在文件中，从文件中读取检测数据，并判断表达式中的括号是否匹配。
 - (2) 1. 将程序模块化，使逻辑更清晰，方便维护。
2. 支持 `()[]{}` 三种括号的匹配，并且可以便捷的扩展更多种括号
3. 支持提示具体位置的括号(和哪一个括号)不匹配
 - (3) 部分逻辑略有些难以梳理；使用 **断点调试** 解决。