|  |
| --- |
| hust1 |
|  |

|  |
| --- |
| **课 程 实 验 报 告** |

|  |
| --- |
| **课程名称： 面向对象程序设计** |
| **实验名称： 面向过程的整形栈编** |

|  |  |
| --- | --- |
| **专业班级：** | **计算机科学与技术201601** |
| **学 号：** | **U201614515`** |
| **姓 名：** | **吴阳民** |
| **指导教师：** | **马光志** |
| **报告日期：** | **2018年 11 月 23 日** |

|  |
| --- |
| **计算机科学与技术学院** |

目 录

[一、需求分析 1](#_Toc529390819)

[1.题目要求 1](#_Toc529390820)

[2.需求分析 1](#_Toc529390821)

[二、系统设计 2](#_Toc529390822)

[1.概要设计 2](#_Toc529390823)

[2.详细设计 2](#_Toc529390824)

[三、软件开发 4](#_Toc529390825)

[四、软件测试 4](#_Toc529390826)

[五、特点与不足 5](#_Toc529390827)

[1.技术特点 5](#_Toc529390828)

[六、遇到的主要问题和解决方法 5](#_Toc529390829)

[七、源码和说明 5](#_Toc529390830)

[1.文件清单及其功能说明 5](#_Toc529390831)

[2.用户使用说明书 6](#_Toc529390832)

[3.源代码 6](#_Toc529390833)

[a.command\_line\_parser.h 6](#_Toc529390834)

[b. command\_line\_parser.cpp 6](#_Toc529390835)

[c. file\_writer.h 8](#_Toc529390836)

[d. file\_writer.cpp 8](#_Toc529390837)

[e. stack\_ex\_1.h 9](#_Toc529390838)

[f. stack\_ex\_1.cpp 10](#_Toc529390839)

[g.U201614515\_1.h 14](#_Toc529390840)

[h. U201614515\_1.cpp 15](#_Toc529390841)

# 一、需求分析

## 1.题目要求

整型栈是一种先进后出的存储结构，对其进行的操作通常包括判断栈是否为空、向栈顶添加一个整型元素、出栈等。整型栈类型及其操作函数采用非面向对象的纯C语言定义，请将完成上述操作的所有函数采用纯C语言编程， 然后写一个main函数对栈的所有操作函数进行测试。

1. **struct** STACK{
2. **int**  \*elems;    //申请内存用于存放栈的元素
3. **int**   max;  //栈能存放的最大元素个数
4. **int**   pos;  //栈实际已有元素个数，栈空时pos=0;
5. };
6. **void** initSTACK(STACK \***const** p, **int** m);  //初始化p指向的栈：最多m个元素
7. **void** initSTACK(STACK \***const** p, **const** STACK&s); //用栈s初始化p指向的栈
8. **int**  size (**const** STACK \***const** p);       //返回p指向的栈的最大元素个数max
9. **int**  howMany (**const** STACK \***const** p);    //返回p指向的栈的实际元素个数pos
10. **int**  getelem (**const** STACK \***const** p, **int** x); //取下标x处的栈元素
11. STACK \***const** push(STACK \***const** p, **int** e);   //将e入栈，并返回p
12. STACK \***const** pop(STACK \***const** p, **int** &e);   //出栈到e，并返回p
13. STACK \***const** assign(STACK\***const** p, **const** STACK&s); //赋s给p指的栈,并返回p
14. **void** print(**const** STACK\***const** p);            //打印p指向的栈
15. **void** destroySTACK(STACK\***const** p);       //销毁p指向的栈

## 2.需求分析

使用C语言的方法实现对栈的基本操作。所要求的操作包括对使用两种方式对栈进行初始化：1.指定栈容量；2.拷贝初始化、计算栈容量、计算栈中元素数量、取栈指定位置元素、入栈、出栈、栈拷贝、栈打印、销毁栈。

# 二、系统设计

## 1.概要设计

用结构体实现栈，该结构体中包含了指向栈的指针，栈的最大容量，当前元素数量。要求实现的栈的类型为整型栈。所以栈指针类型应该为int。

为了能够使用自动测试系统，应该考虑设计命令行解析模块、文件写入模块，保证所有实验都能够使用。栈的操作说明图如图1-1所示。

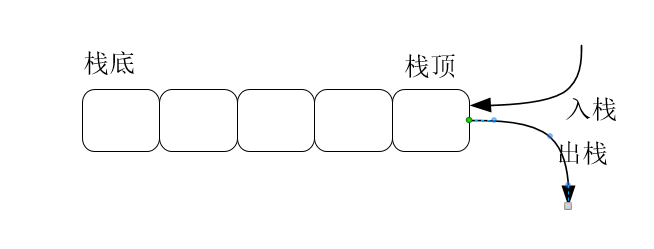


图 1-1

## 2.详细设计

* 通用模块

通用模块是指本次实验以及之后三次实验都能够使用到的相同的模块。

* 命令行解析模块

命令行解析模块用于从命令行参数(argc, argv)中获取信息，并返回给程序合适的输入参数。按照给出的自动测试程序的输入样例，每次输入包含若干命令，每条命令包含若干参数。所以命令行解析程序要能够对输入的参数以及命令进行整理成为一条链表，链表的每个节点包含一条命令与命令所需参数，使得参数与命令能够直接使用。

模块名称：command\_line\_parser

源文件:command\_line\_parser.cpp

头文件:command\_line\_parser.h

结构体: cmd\_list

类名称: COMMAND\_LINE\_PARSER

输入参数: argc, argv

返回值: 一条链表，基本节点为cmd\_list。

* 文件写入模块

为了满足实验测试程序中对程序结果的正确性检查检查要求，将程序运行结果以特定的格式使用文件写入模块写入文件中。

模块名称: file\_writer

源文件：file\_writer.cpp

头文件：file\_writer.h

类名称：FILE\_WRITER

输入参数：char\* file\_name（文件名称）

返回值：无

* 实验一必须模块
* 栈定义模块

模块名称：stack\_ex\_1

源文件：stack\_ex\_1.cpp

头文件：stack\_ex\_1.h

栈结构体：STACK

程序运行入口：run\_stack

* 程序运行模块

模块名称：U201614515\_1

源文件：U201614515\_1.cpp

头文件：U201614515\_1.h

# 三、软件开发

软件开发环境：visual studio Enterprise 2017

软件复测环境：C++ Builder 2009

软件测试工具：checksystem\_v3.0

本地随机测试工具：cmd

# 四、软件测试

正在测试C:/Users/Tortoise/Desktop/U201614515\_1.exe.......

执行命令：C:/Users/Tortoise/Desktop/U201614515\_1.exe -S 5 -I 1 2 3 4 -O 2 -O 2

用户输出:S 5 I 1 2 3 4 O 1 2 O

标准输出:S 5 I 1 2 3 4 O 1 2 O

答案正确！

执行命令：C:/Users/Tortoise/Desktop/U201614515\_1.exe -S 5 -O 0 -I 1 2 3 4 -O 5 -I 1

用户输出:S 5 O I 1 2 3 4 O E

标准输出:S 5 O I 1 2 3 4 O E

答案正确！

执行命令：C:/Users/Tortoise/Desktop/U201614515\_1.exe -S 5 -I 1 2 3 4 -O 2 -I 5 6 7 -I 8

用户输出:S 5 I 1 2 3 4 O 1 2 I 1 2 5 6 7 I E

标准输出:S 5 I 1 2 3 4 O 1 2 I 1 2 5 6 7 I E

答案正确！

执行命令：C:/Users/Tortoise/Desktop/U201614515\_1.exe -S 5 -I 1 2 3 4 -O 2 -A 4 -I 5 6 -I 7 -I 8

用户输出:S 5 I 1 2 3 4 O 1 2 A 1 2 I 1 2 5 6 I 1 2 5 6 7 I E

标准输出:S 5 I 1 2 3 4 O 1 2 A 1 2 I 1 2 5 6 I 1 2 5 6 7 I E

答案正确！

执行命令：C:/Users/Tortoise/Desktop/U201614515\_1.exe -S 5 -I 1 2 3 4 -O 2 -C -I 5 6 -A 2

用户输出:S 5 I 1 2 3 4 O 1 2 C 1 2 I 1 2 5 6 A 1 2 5 6

标准输出:S 5 I 1 2 3 4 O 1 2 C 1 2 I 1 2 5 6 A 1 2 5 6

答案正确！

执行命令：C:/Users/Tortoise/Desktop/U201614515\_1.exe -S 5 -I 1 2 3 4 -O 2 -N

用户输出:S 5 I 1 2 3 4 O 1 2 N 2

标准输出:S 5 I 1 2 3 4 O 1 2 N 2

答案正确！

执行命令：C:/Users/Tortoise/Desktop/U201614515\_1.exe -S 5 -I 1 2 3 4 -G 3 -G 7

用户输出:S 5 I 1 2 3 4 G 4 G E

标准输出:S 5 I 1 2 3 4 G 4 G E

答案正确！

执行命令：C:/Users/Tortoise/Desktop/U201614515\_1.exe -S 5 -I 1 2 3 4 -G 3 -I 5 6 7 8 -O 3 -I 9 0 -G 6 -I 1

用户输出:S 5 I 1 2 3 4 G 4 I E

标准输出:S 5 I 1 2 3 4 G 4 I E

答案正确！

执行命令：C:/Users/Tortoise/Desktop/U201614515\_1.exe -S 3 -I 1 2 3 -O 1 -I 5 6 -G 1 -G 6

用户输出:S 3 I 1 2 3 O 1 2 I E

标准输出:S 3 I 1 2 3 O 1 2 I E

答案正确！

执行命令：C:/Users/Tortoise/Desktop/U201614515\_1.exe -S 3 -I 1 2 3 4 -G 1 -I 5 6 -G 5 -O 6 -O 1

用户输出:S 3 I E

标准输出:S 3 I E

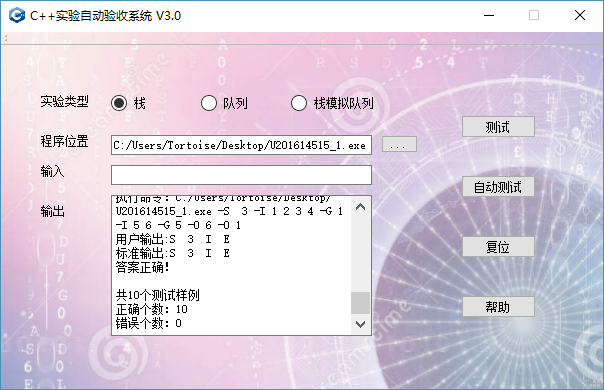
答案正确！

共10个测试样例

正确个数：10

错误个数：0

使用自动测试系统测试结果，如图1-2所示。



**图 1-2**

# 五、特点与不足

## 1.技术特点

各个模块独立编写，在之后的程序中可以直接模块复用。

使用vs开发完成再到c++builder复测，保证本地环境对开发结果无影响。

# 六、遇到的主要问题和解决方法

c++程序多模块编译链接中需要注意不同模块的引用问题，可能导致重复或循环引用，使用预编译指令#pragam once(msvc开发环境下)或#ifndef…#endif(任何环境通用)解决

# 七、源码和说明

## 1.文件清单及其功能说明

* command\_line\_parser.h与command\_line\_parser.cpp为命令行解析模块
* file\_writer.h与file\_writer.cpp为文件写入模块
* stack\_ex\_1.h与stack\_ex\_1.cpp为栈及功能定义模块
* U201614515\_1.h与U201614515\_1.cpp提供程序检测运行入口(main函数)

## 2.用户使用说明书

可执行程序为U201614515\_1.exe程序运行结果保存在U201614515\_1.txt中。源代码在任何环境下可直接建立工程进行编译。

## 3.源代码

### a.command\_line\_parser.h

1. #include<iostream>
2. #include<cstdio>
4. #define COMMAND\_LINE\_PARSER\_H
6. **using** **namespace** std;
8. **typedef** **struct** cmd\_struct {
10. **int**\* elem\_list;
11. **int** size\_elem\_list;
12. **char** cmd;
13. **struct** cmd\_struct \*next;
14. }cmd\_list;
16. **class** COMMAND\_LINE\_PARSER
17. {
18. cmd\_list \* result\_head;
19. **public**:
20. COMMAND\_LINE\_PARSER(**int** argc, **char** \*argv[]);
21. cmd\_list\* get\_list();
22. ~COMMAND\_LINE\_PARSER();
23. };

### b. command\_line\_parser.cpp

1. #include"command\_line\_parser.h"
2. #define max\_cmd\_num 100

5. COMMAND\_LINE\_PARSER::COMMAND\_LINE\_PARSER(**int** args, **char** \*argv[])
6. {
7. result\_head = (cmd\_list\*)malloc(**sizeof**(cmd\_list));
8. cmd\_list\* tail = result\_head;
10. **for** (**int** i = 1; i < args; i++) {
11. **if** (argv[i][0] == '-' && argv[i][1] >= 'A' && argv[i][1] <= 'Z') {
12. tail->next = (cmd\_list\*)malloc(**sizeof**(cmd\_list));
13. tail = tail->next;
14. tail->cmd = argv[i][1];
15. tail->elem\_list = (**int** \*)malloc(**sizeof**(**int**) \* max\_cmd\_num);
16. tail->size\_elem\_list = 0;
17. }
18. **else** {
19. tail->elem\_list[tail->size\_elem\_list++] = atoi(argv[i]);
20. }
21. }
22. tail->next = NULL;
23. tail = result\_head->next;
24. printf("[info] 输入的命令行参数为：\n");
25. **while** (tail != NULL) {
26. printf("%c ", tail->cmd);
27. **for** (**int** i = 0; i < tail->size\_elem\_list; i++) {
28. printf("%d ", tail->elem\_list[i]);
29. }
30. //   printf("\n");
31. tail = tail->next;
32. }
33. printf("[info] 命令行参数解析完毕\n");
34. }
35. cmd\_list \* COMMAND\_LINE\_PARSER::get\_list()
36. {
37. **return** **this**->result\_head;
38. }
40. COMMAND\_LINE\_PARSER::~COMMAND\_LINE\_PARSER()
41. {
42. **while** (result\_head != NULL) {
43. cmd\_list \* tmp = result\_head->next;
44. free(result\_head);
45. result\_head = tmp;
46. }
47. }

### c. file\_writer.h

1. #define FILE\_WRITER\_H
3. #include<string>
4. #include<iostream>
5. #include<fstream>
7. **using** **namespace** std;
8. **class** FILE\_WRITER {
9. **bool** is\_first\_elem;
10. **char**\* file\_name;
11. ofstream\* outfile;
13. **public**:
15. FILE\_WRITER(**char**\* file\_name);
16. **void** writer(**int** num);
17. **void** writer(**char** ch);
18. ~FILE\_WRITER();
19. };

### d. file\_writer.cpp

1. #include "file\_writer.h"
3. FILE\_WRITER::FILE\_WRITER(**char**\* file\_name)
4. {
5. **this**->file\_name = file\_name;
6. **this**->outfile = **new** ofstream;
7. **this**->outfile->open(**this**->file\_name, ios::trunc);
8. **this**->is\_first\_elem = **true**;
9. }
10. **void** FILE\_WRITER::writer(**int** num) {
11. **if** (!**this**->is\_first\_elem) {
12. \*outfile << "  ";
14. \*outfile << num;
15. }**else**{
16. **this**->is\_first\_elem = **false**;
17. \*outfile << num;
18. }

21. }
22. **void** FILE\_WRITER::writer(**char** ch) {
23. **if** (!**this**->is\_first\_elem) {
24. \*outfile << "  ";
25. \*outfile << ch;
26. }**else**{
27. **this**->is\_first\_elem = **false**;
28. \*outfile << ch;
29. }
30. }
32. FILE\_WRITER::~FILE\_WRITER()
33. {
34. **this**->outfile->close();
35. **delete** **this**->outfile;
36. }

### e. stack\_ex\_1.h

1. #pragma once
2. #define STACK\_EX\_1\_H
4. #include<stdio.h>
5. #include<stdlib.h>
6. #include<string>
8. #ifndef COMMAND\_LINE\_PARSER
10. #include"command\_line\_parser.h"
12. #endif // !COMMAND\_LINE\_PARSER
14. #ifndef FILE\_WRITER\_H
16. #include"file\_writer.h"
18. #endif // !FILE\_WRITER\_H


22. #define max\_cmd\_num 100
23. **typedef** **struct** stack {
24. **int**  \*elems;    //申请内存用于存放栈的元素
25. **int**   max;  //栈能存放的最大元素个数
26. **int**   pos;  //栈实际已有元素个数，栈空时pos=0;
27. }STACK;
29. **void** run\_stack(**int** argc, **char** \*\*argv);//测试程序入口
30. **void** initSTACK(STACK \***const** p, **int** m);  //初始化p指向的栈：最多m个元素
31. **void** initSTACK(STACK \***const** p, **const** STACK & s); //用栈s初始化p指向的栈
32. **int**  size(**const** STACK \***const** p);        //返回p指向的栈的最大元素个数max
33. **int**  howMany(**const** STACK \***const** p); //返回p指向的栈的实际    元素个数pos
34. **int**  getelem(**const** STACK \***const** p, **int** x);  //取下标x处的栈元素
35. STACK \***const** push(STACK \***const** p, **int** e);   //将e入栈，并返回p
36. STACK \***const** pop(STACK \***const** p, **int** &e);   //出栈到e，并返回p
37. STACK \***const** assign(STACK\***const** p, **const** STACK&s); //赋s给p指的栈,并返回p
38. **void** print(**const** STACK\***const** p);            //打印p指向的栈
39. **void** destroySTACK(STACK\***const** p);       //销毁p指向的栈

### f. stack\_ex\_1.cpp

1. #include"stack\_ex\_1.h"
3. **bool** cmd\_check\_result = **false**;
4. **char**  file\_path[] = "U201614515\_1.txt";
5. FILE\_WRITER \* file\_writer = **new** FILE\_WRITER(file\_path);
6. **void** run\_stack(**int** argc, **char** \*\*argv) {

9. COMMAND\_LINE\_PARSER \* parser = **new** COMMAND\_LINE\_PARSER(argc, argv);
10. cmd\_list \* cmd\_list\_prase\_result = parser->get\_list();
11. cmd\_list \* tail = cmd\_list\_prase\_result->next;
12. STACK stack\_main;//主栈
13. **while** (tail != NULL) {
15. **switch** (tail->cmd) {
16. **case** 'S':
17. initSTACK(&stack\_main, tail->elem\_list[0]);
18. file\_writer->writer(tail->cmd);
19. //file\_writer(file\_path, tail->cmd);
20. file\_writer->writer(tail->elem\_list[0]);
21. //file\_writer\_num(tail->elem\_list[0]);
22. **break**;
23. **case** 'I':
24. **for** (**int** i = 0; i < tail->size\_elem\_list; i++) {
25. push(&stack\_main, tail->elem\_list[i]);
26. }
27. file\_writer->writer(tail->cmd);
28. **if** (cmd\_check\_result == **true**) {
29. file\_writer->writer('E');
30. **return**;
31. }
32. print(&stack\_main);
33. **break**;
34. **case** 'O':
35. **int** e;
36. file\_writer->writer(tail->cmd);
37. **for** (**int** i = 0; i < tail->elem\_list[0]; i++) {
38. pop(&stack\_main, e);
39. }
41. **if** (cmd\_check\_result == **true**) {
42. file\_writer->writer('E');
43. **return**;
44. }
46. print(&stack\_main);
47. **break**;
48. **case** 'C':
49. assign(&stack\_main, stack\_main);
51. file\_writer->writer(tail->cmd);
52. print(&stack\_main);
53. **break**;
54. **case** 'A':
55. assign(&stack\_main, stack\_main);
56. file\_writer->writer(tail->cmd);
57. print(&stack\_main);
58. **break**;
59. **case** 'N':
60. file\_writer->writer(tail->cmd);
61. file\_writer->writer(howMany(&stack\_main));
62. **break**;
63. **case** 'G':
64. file\_writer->writer(tail->cmd);
65. **if** (tail->elem\_list[0] >= howMany(&stack\_main)) {
66. file\_writer->writer('E');
67. **return**;
68. }
69. **else** {
70. file\_writer->writer(getelem(&stack\_main, tail->elem\_list[0]));
71. //file\_writer\_num(getelem(&stack\_main, tail->elem\_list[0]));
72. }
73. **break**;
74. **default**:
75. **break**;
76. }
77. tail = tail->next;
78. }
79. **delete** file\_writer;
80. }
82. **void** initSTACK(STACK \***const** p, **int** m) {//初始化p指向的栈：最多m个元素
83. **if** (m <= 0) {
84. printf("[error] 不能初始化空间大小为%d的栈!\n", m);
85. **return**;
86. }
87. p->elems = (**int** \*)malloc(**sizeof**(**int**) \* m);
88. p->max = m;
89. p->pos = 0;
90. }
91. **void** initSTACK(STACK \***const** p, **const** STACK &s) {//用栈s初始化p指向的栈
92. **if** (s.max == 0) {
93. printf("[error]  栈s为空,请初始化！\n");
94. **return**;
95. }
96. initSTACK(p, s.max);
97. **for** (**int** i = 0; i < s.pos; i++) {
98. \*(p->elems + i) = \*(s.elems + i);
99. ++(p->pos);
100. }
101. }
102. **int**  size(**const** STACK \***const** p) {//返回p指向的栈的最大元素个数max
103. **return** p->max;
104. }
105. **int**  howMany(**const** STACK \***const** p) {//返回p指向的栈的实际元素个数pos
106. **return** p->pos;
107. }
108. **int**  getelem(**const** STACK \***const** p, **int** x) {//取下标x处的栈元素
109. **if** (x < 0) {
110. printf("[error] 不能获取位置%d处的栈元素!\n", x);
111. **return** 0;
112. }
113. **return** (p->elems)[x];
114. }
116. STACK \***const** push(STACK \***const** p, **int** e) {//将e入栈，并返回p
117. **if** (howMany(p) == size(p)) {
118. printf("[error] 栈已满！\n");
119. cmd\_check\_result = **true**;
120. **return** p;
121. }
122. **if** (size(p) == 0) {
123. printf("[error] 栈大小为0！\n");
124. cmd\_check\_result = **true**;
125. **return** p;
126. }
127. p->elems[p->pos++] = e;
128. **return** p;
129. }
131. STACK \***const** pop(STACK \***const** p, **int** &e) {//出栈到e，并返回p
132. **if** (howMany(p) == 0) {
133. printf("[erroe] 栈为空！\n");
134. cmd\_check\_result = **true**;
135. **return** p;
136. }
137. e = (p->elems)[p->pos--];
138. **return** p;
139. }
141. STACK \***const** assign(STACK\***const** p, **const** STACK&s) {//赋s给p指的栈,并返回p
142. p->elems = s.elems;
143. p->max = s.max;
144. p->pos = s.pos;
145. **return** p;
147. }
148. **void** print(**const** STACK\***const** p) {           //打印p指向的栈
149. printf("[elem in stack]  [");
150. **for** (**int** i = 0; i < p->pos; i++) {
151. printf("%d", p->elems[i]);
152. **if** (i != p->pos - 1) {
153. printf(", ");
154. }
155. }
156. printf("]\n");
157. **for** (**int** i = 0; i < p->pos; i++) {
158. file\_writer->writer(p->elems[i]);
159. //file\_writer\_num(p->elems[i]);
160. }
161. **return**;
162. }
163. **void** destroySTACK(STACK\***const** p) {      //销毁p指向的栈
164. free(p->elems);
165. p->max = 0;
166. p->pos = 0;
167. }

### g.U201614515\_1.h

1. #define \_CRT\_SECURE\_NO\_WARNINGS
2. #pragma once
3. #define RUN\_EX\_1
4. //#define RUN\_EX\_2
5. //#define RUN\_EX\_3
6. //#define RUN\_EX\_6
8. #ifndef STACK\_EX\_1\_H
9. #ifdef RUN\_EX\_1
10. #include"stack\_ex\_1.h"
11. #endif // RUN\_EX\_1
12. #endif // !STACK\_EX\_1\_H

15. #ifndef STACK\_OPERATOR\_OVERLOADING\_H
16. #ifdef RUN\_EX\_2
17. #include"stack\_operator\_overloading.h"
18. #endif // RUN\_EX\_2
19. #endif // !STACK\_OPERATOR\_OVERLOADING\_H

22. #ifndef QUEUE\_EX\_3\_H
23. #ifdef RUN\_EX\_3
24. #include"queue\_ex\_3.h"
25. #endif // RUN\_EX\_3
26. #endif // !QUEUE\_EX\_3\_H
28. #ifndef QUEUE\_EX\_6
29. #ifdef RUN\_EX\_6
30. #include"queue\_ex\_6.h"
31. #endif // RUN\_EX\_6
32. #endif // !QUEUE\_EX\_6
33. **using** **namespace** std;

### h. U201614515\_1.cpp

1. #include"U201614515\_1.h"
2. #define RUN\_FUNCTION
3. **int** main(**int** argc, **char**\* argv[]) {
4. //运行栈检测
5. run\_stack(argc, argv);
6. //run\_stack\_2(argc, argv);
7. //run\_queue\_ex\_3(argc, argv);
8. //run\_queue\_ex\_6(argc, argv);
9. //system("pause");
10. **return** 0;
11. }