|  |
| --- |
| 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程** |

|  |  |
| --- | --- |
| **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](#_Toc530407251)

[1.题目要求 1](#_Toc530407252)

[2.需求分析 2](#_Toc530407253)

[二、系统设计 2](#_Toc530407254)

[1.概要设计 2](#_Toc530407255)

[2.详细设计 2](#_Toc530407256)

[a.栈的构造 2](#_Toc530407257)

[b.栈容量获取 2](#_Toc530407258)

[c.获取当前元素个数 3](#_Toc530407259)

[d.获取指定位置元素 3](#_Toc530407260)

[e.入栈 4](#_Toc530407261)

[f.出栈 4](#_Toc530407262)

[g.复制 4](#_Toc530407263)

[h.打印 5](#_Toc530407264)

[三、软件开发 5](#_Toc530407265)

[四、软件测试 5](#_Toc530407266)

[五、特点与不足 8](#_Toc530407267)

[1.技术特点 8](#_Toc530407268)

[2.不足 8](#_Toc530407269)

[七、源码和说明 8](#_Toc530407270)

[1.文件清单及其功能说明 8](#_Toc530407271)

[2.用户使用说明书 8](#_Toc530407272)

[3.源代码 9](#_Toc530407273)

[a. stack\_operator\_overloading.h 9](#_Toc530407274)

[b. stack\_operator\_overloading.cpp 10](#_Toc530407275)

[c.U201614515\_2.h 13](#_Toc530407276)

[d.U201614515\_2.cpp 14](#_Toc530407277)

# 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面向对象的C++语言定义，请将完成上述操作的所有函数采用C++编程， 然后写一个main函数对栈的所有操作函数进行测试。

1. **class** STACK{
2. **int**  \***const**  elems; //申请内存用于存放栈的元素
3. **const**  **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. **public**:
6. STACK(**int** m);       //初始化栈：最多m个元素
7. STACK(**const** STACK&s); //用栈s拷贝初始化栈
8. **int**  size ( ) **const**;        //返回栈的最大元素个数max
9. **int**  howMany ( ) **const**; //返回栈的实际元素个数pos
10. **int**  getelem (**int** x) **const**; //取下标x处的栈元素
11. STACK& push(**int** e);     //将e入栈,并返回栈
12. STACK& pop(**int** &e);     //出栈到e,并返回栈
13. STACK& assign(**const** STACK&s); //赋s给栈,并返回被赋值的栈
14. **void** print( ) **const**;        //打印栈
15. ~STACK( );              //销毁栈
16. };

注意：虽然实现的队列 最多能入2m个元素，但是由于模拟中入出操作序列的限制，即操作不能违背先进先出的原则，可能还没入2m个元素队列就“满”了，所以要注意full( )函数的实现：不能简单判断队列是否装了2m个元素。

## 2.需求分析

设计面向对象的C++语言定义整型栈及基本操作，栈操作包括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

用类实现栈。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。

使用第一次实验中设计的文件写入模块与命令行解析模块。

## 2.详细设计

### a.栈的构造

方法名称：STACK

入口参数：m代表栈初始容量

返回值：无

出口参数：无

函数功能：构造函数，分配栈空间，初始化栈相关参数



### b.栈容量获取

方法名称：size

入口参数：无

返回值：栈容量（对象属性max的值）

出口参数：无

函数功能：返回栈容量



### c.获取当前元素个数

方法名称：重载运算符()

入口参数：无

返回值：当前栈中元素数量（对象属性pos的值）

出口参数：无



### d.获取指定位置元素

方法名称：重载运算符[]

入口参数：int x（栈中x位置）

返回值：给定下标处栈中的元素

出口参数：下标为x的栈中元素



### e.入栈

方法名称：重载运算符<<

入口参数：int e（需要入栈的元素）

出口参数：this

函数功能：入栈



### f.出栈

方法名称：重载运算符>>

入口参数：int &e

出口参数：this

函数功能：将栈顶元素出栈到e



### g.复制

方法名称：重载运算符=

入口参数：待复制的栈的引用

出口参数：this

函数功能：将s指向this指向的栈



### h.打印

函数名称：print

入口参数：无

出口参数：无

返回值：this

函数功能：打印栈中现有元素



# 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\_3.exe.......

执行命令：C:/Users/Tortoise/Desktop/U201614515\_3.exe -S 5 -I 1 2 3 4 -O 2 -O 2

用户输出:S 5 I 1 2 3 4 O 3 4 O

标准输出:S 5 I 1 2 3 4 O 3 4 O

答案正确！

执行命令：C:/Users/Tortoise/Desktop/U201614515\_3.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\_3.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 3 4 I 3 4 5 6 7 I 3 4 5 6 7 8

标准输出:S 5 I 1 2 3 4 O 3 4 I 3 4 5 6 7 I 3 4 5 6 7 8

答案正确！

执行命令：C:/Users/Tortoise/Desktop/U201614515\_3.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 3 4 A 3 4 I 3 4 5 6 I 3 4 5 6 7 I 3 4 5 6 7 8

标准输出:S 5 I 1 2 3 4 O 3 4 A 3 4 I 3 4 5 6 I 3 4 5 6 7 I 3 4 5 6 7 8

答案正确！

执行命令：C:/Users/Tortoise/Desktop/U201614515\_3.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 3 4 C 3 4 I 3 4 5 6 A 3 4 5 6

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

答案正确！

执行命令：C:/Users/Tortoise/Desktop/U201614515\_3.exe -S 5 -I 1 2 3 4 -O 2 -N

用户输出:S 5 I 1 2 3 4 O 3 4 N 2

标准输出:S 5 I 1 2 3 4 O 3 4 N 2

答案正确！

执行命令：C:/Users/Tortoise/Desktop/U201614515\_3.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\_3.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 1 2 3 4 5 6 7 8 O 4 5 6 7 8 I 4 5 6 7 8 9 0 G 0 I E

标准输出:S 5 I 1 2 3 4 G 4 I 1 2 3 4 5 6 7 8 O 4 5 6 7 8 I 4 5 6 7 8 9 0 G 0 I E

答案正确！

执行命令：C:/Users/Tortoise/Desktop/U201614515\_3.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 2 3 I 2 3 5 6 G 3 G E

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

答案正确！

执行命令：C:/Users/Tortoise/Desktop/U201614515\_3.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 1 2 3 4 G 2 I 1 2 3 4 5 6 G 6 O O E

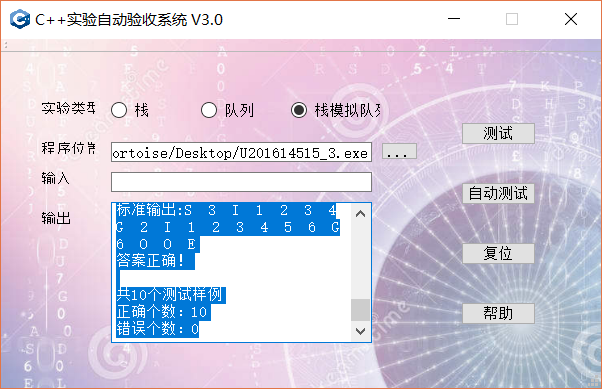
标准输出:S 3 I 1 2 3 4 G 2 I 1 2 3 4 5 6 G 6 O O E

答案正确！

共10个测试样例

正确个数：10

错误个数：0



# 五、特点与不足

## 1.技术特点

无

## 2.不足

无

# 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\_operator\_overloading.h与stack\_operator\_overloading.cpp为栈及功能定义模块
* U201614515\_2.h与U201614515\_2.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\_2.exe程序运行结果保存在U201614515\_2.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. stack\_operator\_overloading.h

1. #pragma once
2. #include<iostream>
4. #ifndef COMMAND\_LINE\_PARSER\_H
6. #include"command\_line\_parser.h"
8. #endif // !COMMAND\_LINE\_PARSER\_H
10. #ifndef FILE\_WRITER\_H
12. #include"file\_writer.h"
14. #endif // !FILE\_WRITER\_H
16. #define STACK\_OPERATOR\_OVERLOADING\_H
18. **using** **namespace** std;
19. **class** STACK {
20. **int**  \***const** elems;  //申请内存用于存放栈的元素
21. **const** **int**   max;    //栈能存放的最大元素个数
22. **int**   pos;          //栈实际已有元素个数，栈空时pos=0;
23. **public**:
24. FILE\_WRITER \* file\_writer;
25. **bool** error\_flag;
26. STACK(**int** m);       //初始化栈：最多存m个元素
27. STACK(**const** STACK&s);           //用栈s拷贝初始化栈
28. **int**  size() **const**;          //返回栈的最大元素个数max
29. **int**  howMany ( ) **const**; //返回栈的实际元素个数pos
30. **int**  getElem (**int** x) **const**; //取下标x处的栈元素
31. STACK& push(**int** e);     //将e入栈,并返回栈
32. STACK& pop(**int** &e);     //出栈到e,并返回栈
33. STACK& assign(**const** STACK&s); //赋s给栈,并返回被赋值的栈
34. **void** print() **const**;         //打印栈
35. ~STACK();                   //销毁栈
36. };
38. **void** run\_stack\_2(**int** argc, **char**\* argv[]);

### b. stack\_operator\_overloading.cpp

1. #include"stack\_operator\_overloading.h"

4. **bool** cmd\_check = **false**;
5. STACK::STACK(**int** m):max(m), elems(**new** **int**[m]) {     //初始化栈：最多存m个元素
6. **this**->pos = 0;
7. }
8. STACK::STACK(**const** STACK&s):max(s.max), elems(s.elems) {            //用栈s拷贝初始化栈
9. **this**->pos = s.pos;
10. }
11. **int** STACK::size() **const** {           //返回栈的最大元素个数max
12. **return** **this**->max;
13. }
14. **int** STACK::howMany()**const** {//返回当前元素个数
15. **return** **this**->pos;
16. }
17. **int** STACK::getElem(**int** x)**const** {    //取下标x处的栈元素，第1个元素x=0
18. **if** (x >= **this**->pos) {
19. cout << "[error] 栈中有" << **this**->pos << "个元素，无法取出位置为" << x << "的元素";
20. cmd\_check = **true**;
21. **return** 0;
22. }
23. **return** **this**->elems[x];
24. }
25. STACK &STACK::push(**int** e) {     //将e入栈,并返回栈
26. **if** (**this**->pos >= **this**->size()) {
27. cout << "[error] 栈已满,入栈失败" << endl;
28. cmd\_check = **true**;
29. **return** \***this**;
30. }
31. **this**->elems[**this**->pos++] = e;
32. **return** \***this**;
33. }
34. STACK &STACK::pop(**int** &e) { //出栈到e,并返回栈
35. **if** (**this**->pos <= 0) {
36. e = 0;
37. **return** \***this**;
38. }
39. e = **this**->elems[--**this**->pos];
40. **return** \***this**;
41. }
42. STACK &STACK::assign(**const** STACK &s){ //赋s给栈,并返回被赋值的栈
43. **delete**[] **this**->elems;
44. \*((**int** \*\*)&elems) = **new** **int**[s.max];
45. \*((**int** \*)&(**this**->max))=s.max;
46. **for**(**int** i = 0;i <s.pos;i++){
47. **this**->elems[i] = s.elems[i];
48. }
49. **this**->pos = s.pos;
50. **return** \***this**;
51. }
52. **void**  STACK::print() **const** {            //打印栈
53. cout << "[info] stack elme to string:[";
54. **for** (**int** i = 0; i < **this**->pos; i++) {
55. cout << **this**->elems[i];
56. **if** (i != **this**->pos - 1) {
57. cout << ", ";
58. }
59. **else** {
60. cout << "]" << endl;
61. }
62. }
63. **for** (**int** i = 0; i < **this**->pos; i++) {
64. **this**->file\_writer->writer(**this**->elems[i]);
65. }
66. }
67. STACK::~STACK() {
68. **delete** **this**->elems;
69. **this**->pos = 0;
70. }
72. **void** run\_stack\_2(**int** argc, **char**\* argv[]) {
73. **char** file\_name[] = "U201614515\_3.txt";
74. COMMAND\_LINE\_PARSER \* command\_line\_parser = **new** COMMAND\_LINE\_PARSER(argc, argv);
75. cmd\_list \* head = command\_line\_parser->get\_list();
76. cmd\_list \*tail = head->next;
77. STACK\* stack = **new** STACK(10);
78. **while** (tail != NULL) {
79. **switch** (tail->cmd) {
80. **case** 'S':
81. stack = **new** STACK(tail->elem\_list[0]);
82. stack->file\_writer = **new** FILE\_WRITER(file\_name);
83. stack->file\_writer->writer(tail->cmd);
84. printf("%c  ", tail->cmd);
85. stack->file\_writer->writer(tail->elem\_list[0]);
86. printf("%d  ", tail->elem\_list[0]);
87. **break**;
88. **case** 'I':
89. **for** (**int** i = 0; i < tail->size\_elem\_list; i++) {
90. (\*stack).push(tail->elem\_list[i]);
91. }
92. stack->file\_writer->writer(tail->cmd);
93. printf("%c  ", tail->cmd);
94. **if** (cmd\_check == **true**) {
95. stack->file\_writer->writer('E');
96. printf("%c  ", 'E');
97. **return**;
98. }
99. stack->print();
100. **break**;
101. **case** 'O':
102. **int** e;
103. printf("option O point hit\n");
104. stack->file\_writer->writer(tail->cmd);
105. printf("%c  ", tail->cmd);
106. printf("option O point hit par: size = %d\n", tail->size\_elem\_list);
108. **if** (tail->size\_elem\_list == 0 || tail->elem\_list[0] > stack->howMany()) {
109. cmd\_check = **true**;
110. }
111. **else** {
113. **for** (**int** i = 0; i < tail->elem\_list[0]; i++) {
114. stack->pop(e);
115. }
116. }
117. **if** (cmd\_check == **true**) {
118. stack->file\_writer->writer('E');
119. printf("%c ", 'E');
120. **return**;
121. }
123. stack->print();
124. **break**;
125. **case** 'C':
126. stack->file\_writer->writer(tail->cmd);
127. printf("%c  ", tail->cmd);
128. stack->print();
129. **break**;
130. **case** 'A':
131. stack->file\_writer->writer(tail->cmd);
132. printf("%c  ", tail->cmd);
133. stack->print();
134. **break**;
135. **case** 'N':
136. stack->file\_writer->writer(tail->cmd);
137. printf("%c  ", tail->cmd);
138. stack->file\_writer->writer(stack->howMany());
139. //\_num(howMany(&stack\_main));
140. **break**;
141. **case** 'G':
142. stack->file\_writer->writer(tail->cmd);
143. printf("%c  ", tail->cmd);
144. **if** (tail->elem\_list[0] >= stack->howMany()) {
145. stack->file\_writer->writer('E');
146. printf("%c  ", 'E');
147. **return**;
148. }
149. **else** {
150. stack->file\_writer->writer(stack->getElem(tail->elem\_list[0]));
151. }
152. **break**;
153. **default**:
154. **break**;
155. }
156. tail = tail->next;
157. }
158. **delete** stack->file\_writer;
159. //system("pause");
160. }

### c.U201614515\_2.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
34. **using** **namespace** std;

### d.U201614515\_2.cpp

1. #include"U201614515\_3.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. }