|  |
| --- |
| hust1 |

|  |
| --- |
| **课 程 实 验 报 告** |

|  |
| --- |
| **课程名称： 面向对象程序设计** |
| **实验名称： 整型栈运算符重载编程** |

|  |  |
| --- | --- |
| **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](#_Toc529734622)

[1.题目要求 1](#_Toc529734623)

[2.需求分析 1](#_Toc529734624)

[二、系统设计 2](#_Toc529734625)

[1.概要设计 2](#_Toc529734626)

[2.详细设计 2](#_Toc529734627)

[三、软件开发 3](#_Toc529734628)

[四、软件测试 4](#_Toc529734629)

[五、特点与不足 6](#_Toc529734630)

[1.技术特点 6](#_Toc529734631)

[2.不足 6](#_Toc529734632)

[七、源码和说明 6](#_Toc529734633)

[1.文件清单及其功能说明 6](#_Toc529734634)

[2.用户使用说明书 6](#_Toc529734635)

[3.源代码 7](#_Toc529734636)

[a. stack\_operator\_overloading.h 7](#_Toc529734637)

[b. stack\_operator\_overloading.cpp 8](#_Toc529734638)

[c.U201614515\_2.h 12](#_Toc529734639)

[d.U201614515\_2.cpp 12](#_Toc529734640)

# 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：最多存m个元素
7. STACK(**const** STACK&s);           //用栈s拷贝初始化栈
8. **virtual** **int**  size ( ) **const**;            //返回栈的最大元素个数max
9. **virtual** operator **int** ( ) **const**;         //返回栈的实际元素个数pos
10. **virtual** **int** operator[ ] (**int** x) **const**;  //取下标x处的栈元素，第1个元素x=0
11. **virtual** STACK& operator<<(**int** e);     //将e入栈,并返回栈
12. **virtual** STACK& operator>>(**int** &e);    //出栈到e,并返回栈
13. **virtual** STACK& operator=(**const** STACK&s); //赋s给栈,并返回被赋值的栈
14. **virtual** **void** print( ) **const**;            //打印栈
15. **virtual** ~STACK( );                  //销毁栈
16. };

## 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

a.虚函数

在面向对象程序设计中，派生类继承自基类。使用[指针](https://zh.wikipedia.org/wiki/%E6%8C%87%E9%92%88_(%E4%BF%A1%E6%81%AF%E5%AD%A6))或[引用](https://zh.wikipedia.org/wiki/%E5%8F%83%E7%85%A7)访问派生类对象时，指针或引用本身所指向的类型可以是基类而不是派生类。如果派生类覆盖了基类中的方法，通过上述指针或引用调用该方法时，可以有两种结果：

调用到基类的方法：编译器根据指针或引用的类型决定，称作“早绑定”；

调用到派生类的方法：语言的运行时系统根据对象的实际类型决定，称作“迟绑定”。

虚函数的效果属于后者。如果问题中基类的函数是“虚”的，则调用到的都是最终派生类（英语：most-derived class）中的函数实现，与指针或引用的类型无关。反之，如果函数非“虚”，调用到的函数就在编译期根据指针或者引用所指向的类型决定。

有了虚函数，程序甚至能够调用编译期还不存在的函数。

在[C++](https://zh.wikipedia.org/wiki/C++)中，在基类的成员函数声明前加上关键字 virtual 即可让该函数成为虚函数，派生类中对此函数的不同实现都会继承这一修饰符，允许后续派生类覆盖，达到迟绑定的效果。即便是基类中的成员函数调用虚函数，也会调用到派生类中的版本。

### b.符号重载

[C++](https://wiki.mbalib.com/wiki/C++)有许多内置的数据类型，包括int，char，double等，每一种类型都有许多运算符，例如加，减，乘，除等。当用户定义了类的对象时，两个对象之间是不能进行这些操作的，比如类的对象a+b，这样的语句如果没有重载+运算符就会出错。但C++允许用户把这些运算符添加到自已的类中以方便类对象之间的运算就像内置类型的运算一样方便，比如对象a+b这样就很明白更容易懂，当然也可以在类中定义一个对象间相加的函数，比如a.add(b)调用函数add()以实现两个对象a和b相加，但是这条语句没有比a+b更容易让人理解。

## 2.详细设计

1. **class** 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. **public**:
6. FILE\_WRITER \* file\_writer;
7. **bool** error\_flag;
8. STACK(**int** m);       //初始化栈：最多存m个元素
9. STACK(**const** STACK&s);           //用栈s拷贝初始化栈
10. **virtual** **int**  size() **const**;          //返回栈的最大元素个数max
11. **virtual** operator **int**() **const**;       //返回栈的实际元素个数pos
12. **virtual** **int** operator[] (**int** x) **const**;   //取下标x处的栈元素，第1个元素x=0
13. **virtual** STACK& operator<<(**int** e);     //将e入栈,并返回栈
14. **virtual** STACK& operator>>(**int** &e);    //出栈到e,并返回栈
15. **virtual** STACK& operator=(**const** STACK&s); //赋s给栈,并返回被赋值的栈
16. **virtual** **void** print() **const**;         //打印栈
17. **virtual** ~STACK();                   //销毁栈
18. };

a.类成员变量

int \*elems，用于指向存储元素的存储空间的指针。

int max，指示栈中最多存储的元素数量。

int pos，指示当前栈中存储的元素数量，也是下一个元素可以存放的位置。

b.类成员函数

virtual int size() const 获取栈所能存储的最大元素数量，操作结果是成员变量max的值。

virtual operator int() const 获取栈当前存储的元素shuliang ，操作结果是成员变量pos的值。

virtual int operator[] (int x) const 提供以直观的数组方式访问栈的方法。

virtual STACK& operator<<(int e) 重载<<运算符，提供入栈功能，返回值为当前栈的指针。

virtual STACK& operator>>(int &e) 重载>>运算符，提供出栈狗跟那个，返回值为当前栈的指针.

virtual STACK& operator=(const STACK&s) 提供以另一个类初始化当前栈的方法，返回值为当前栈。

virtual void print() const 提供打印栈的方法。

# 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\_2.exe.......

执行命令：C:/Users/Tortoise/Desktop/U201614515\_2.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\_2.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\_2.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\_2.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\_2.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\_2.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\_2.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\_2.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\_2.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\_2.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

使用自动测试软件测试结果如图2-1所示。

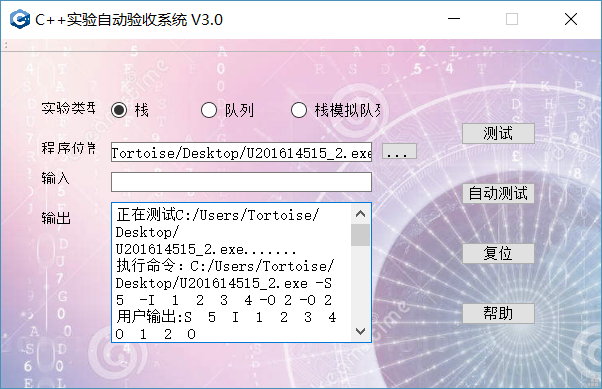


图 2-1

# 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。

## 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**  \*elems;    //申请内存用于存放栈的元素
21. **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. **virtual** **int**  size() **const**;          //返回栈的最大元素个数max
29. **virtual** operator **int**() **const**;       //返回栈的实际元素个数pos
30. **virtual** **int** operator[] (**int** x) **const**;   //取下标x处的栈元素，第1个元素x=0
31. **virtual** STACK& operator<<(**int** e);     //将e入栈,并返回栈
32. **virtual** STACK& operator>>(**int** &e);    //出栈到e,并返回栈
33. **virtual** STACK& operator=(**const** STACK&s); //赋s给栈,并返回被赋值的栈
34. **virtual** **void** print() **const**;         //打印栈
35. **virtual** ~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) {       //初始化栈：最多存m个元素
6. **this**->elems = **new** **int**[m];
7. **this**->max = m;
8. **this**->pos = 0;
9. }
10. STACK::STACK(**const** STACK&s) {           //用栈s拷贝初始化栈
11. **this**->elems = **new** **int**[s.size()];
12. **for** (**int** i = 0; i < **int**(s); i++) {
13. **this**->elems[i] = s.elems[i];
14. }
15. **this**->max = s.max;
16. **this**->pos = s.pos;
17. }
18. **int** STACK::size() **const** {           //返回栈的最大元素个数max
19. **return** **this**->max;
20. }
21. STACK::operator **int**() **const**
22. {
23. **return** **this**->pos;
24. }
25. **int** STACK::operator[] (**int** x) **const** {   //取下标x处的栈元素，第1个元素x=0
26. **if** (x >= **this**->pos) {
27. cout << "[error] 栈中有" << **this**->pos << "个元素，无法取出位置为" << x << "的元素";
28. cmd\_check = **true**;
29. **return** 0;
30. }
31. **return** **this**->elems[x];
32. }
33. STACK& STACK::operator<<(**int** e) {     //将e入栈,并返回栈
34. **if** (**this**->pos >= **this**->size()) {
35. cout << "[error] 栈已满,入栈失败" << endl;
36. cmd\_check = **true**;
37. **return** \***this**;
38. }
39. **this**->elems[**this**->pos++] = e;
40. **return** \***this**;
41. }
42. STACK& STACK::operator>>(**int** &e) {    //出栈到e,并返回栈
43. **if** (**this**->pos <= 0) {
44. e = 0;
45. **return** \***this**;
46. }
47. e = **this**->elems[--**this**->pos];
48. **return** \***this**;
49. }
50. STACK& STACK::operator=(**const** STACK&s) { //赋s给栈,并返回被赋值的栈
51. **this**->elems = s.elems;
52. **this**->max = s.max;
53. **this**->pos = s.pos;
54. **return** \***this**;
55. }
56. **void**  STACK::print() **const** {            //打印栈
57. cout << "[info] stack elme to string:[";
58. **for** (**int** i = 0; i < **this**->pos; i++) {
59. cout << **this**->elems[i];
60. **if** (i != **this**->pos - 1) {
61. cout << ", ";
62. }
63. **else** {
64. cout << "]" << endl;
65. }
66. }
67. **for** (**int** i = 0; i < **this**->pos; i++) {
68. **this**->file\_writer->writer(**this**->elems[i]);
69. }
70. }
71. STACK::~STACK() {
72. **delete** **this**->elems;
73. **this**->pos = 0;
74. **this**->max = 0;
75. }
77. **void** run\_stack\_2(**int** argc, **char**\* argv[]) {
78. **char** file\_name[] = "U201614515\_2.txt";
79. COMMAND\_LINE\_PARSER \* command\_line\_parser = **new** COMMAND\_LINE\_PARSER(argc, argv);
80. cmd\_list \* head = command\_line\_parser->get\_list();
81. cmd\_list \*tail = head->next;
82. STACK\* stack = **new** STACK(10);
83. **while** (tail != NULL) {
84. **switch** (tail->cmd) {
85. **case** 'S':
86. stack = **new** STACK(tail->elem\_list[0]);
87. stack->file\_writer = **new** FILE\_WRITER(file\_name);
88. stack->file\_writer->writer(tail->cmd);
89. printf("%c  ", tail->cmd);
90. stack->file\_writer->writer(tail->elem\_list[0]);
91. printf("%d  ", tail->elem\_list[0]);
92. **break**;
93. **case** 'I':
94. **for** (**int** i = 0; i < tail->size\_elem\_list; i++) {
95. (\*stack) << tail->elem\_list[i];
96. }
97. stack->file\_writer->writer(tail->cmd);
98. printf("%c  ", tail->cmd);
99. **if** (cmd\_check == **true**) {
100. stack->file\_writer->writer('E');
101. printf("%c  ", 'E');
102. **return**;
103. }
104. stack->print();
105. **break**;
106. **case** 'O':
107. **int** e;
108. printf("option O point hit\n");
109. stack->file\_writer->writer(tail->cmd);
110. printf("%c  ", tail->cmd);
111. printf("option O point hit par: size = %d\n", tail->size\_elem\_list);
113. **if** (tail->size\_elem\_list == 0 || tail->elem\_list[0] > **int**(\*stack)) {
114. cmd\_check = **true**;
115. }
116. **else** {
118. **for** (**int** i = 0; i < tail->elem\_list[0]; i++) {
119. (\*stack) >> e;
120. }
121. }
122. **if** (cmd\_check == **true**) {
123. stack->file\_writer->writer('E');
124. printf("%c ", 'E');
125. **return**;
126. }
128. stack->print();
129. **break**;
130. **case** 'C':
131. stack->file\_writer->writer(tail->cmd);
132. printf("%c  ", tail->cmd);
133. stack->print();
134. **break**;
135. **case** 'A':
136. stack->file\_writer->writer(tail->cmd);
137. printf("%c  ", tail->cmd);
138. stack->print();
139. **break**;
140. **case** 'N':
141. stack->file\_writer->writer(tail->cmd);
142. printf("%c  ", tail->cmd);
143. stack->file\_writer->writer(**int**(\*stack));
144. //\_num(howMany(&stack\_main));
145. **break**;
146. **case** 'G':
147. stack->file\_writer->writer(tail->cmd);
148. printf("%c  ", tail->cmd);
149. **if** (tail->elem\_list[0] >= **int**(\*stack)) {
150. stack->file\_writer->writer('E');
151. printf("%c  ", 'E');
152. **return**;
153. }
154. **else** {
155. stack->file\_writer->writer((\*stack)[tail->elem\_list[0]]);
156. }
157. **break**;
158. **default**:
159. **break**;
160. }
161. tail = tail->next;
162. }
163. **delete** stack->file\_writer;
164. //system("pause");
165. }

### 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\_2.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. }