

华中科技大学

# 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

院 系：计算机科学与技术

专业班级：物联网工程 1601

学 号：U201614898

姓 名：潘 翔

指导教师：

2018 年 10 月 16 日

## 1 需求分析

### 1.1 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。要求 main 按照《关于 C++ 实验自动验收系统说明》给定的方式工作。

```
class STACK
{
    int *const elems; //申请内存用于存放栈的元素
    const int max; //栈能存放的最大元素个数
    int pos; //栈实际已有元素个数，栈空时 pos=0;
public:
    STACK(int m); //初始化栈：最多存 m 个元素
    STACK(const STACK&s); //用栈 s 拷贝初始化栈
    virtual int size() const; //返回栈的最大元素个数 max
    virtual operator int() const; //返回栈的实际元素个数 pos
    virtual int operator[] (int x) const; //取下标 x 处的栈元素，第 1 个元素 x=0
    virtual STACK& operator<<(int e); //将 e 入栈,并返回栈
    virtual STACK& operator>>(int &e); //出栈到 e,并返回栈
    virtual STACK& operator=(const STACK&s); //赋 s 给栈,并返回被赋值的栈
    virtual void print() const; //打印栈
    virtual ~STACK(); //销毁栈
};
```

### 1.2 需求分析

利用 C++ 语言实现整型栈，栈是一种具有后入先出(LIFO)性质的线性数据结构，此次实验要求完成在栈上的一些基本操作，具体包括构造、取大小、取元素个数、下标访问元素、入栈、出栈、赋值、相等判定(自加功能)、(打印)输出、销毁等操作。

## 2 系统设计

### 2.1 概要设计

介绍设计思路、原理。将一个复杂系统按功能进行模块划分、建立模块的层次结构及调用关系、确定模块间的接口及人机界面等。

要有总体结构、总体流程（图）。

#### 2.2.1 系统整体架构图

本系统分为 4 个模块：平台判断模块，mian 调用模块，数据结构模块，调试模块。

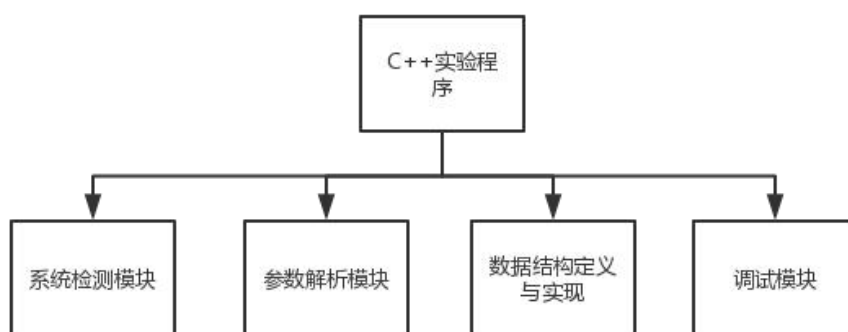


图 1-1 系统整体架构图

STACK
- elems : int* const
- max : const int
- pos : int
+ setpos(v : int)
+ getpos() : int
+ STACK(m : int) «constructor»
+ STACK(s : const STACK&) «constructor»
+ size() : int
+ full() : bool
+ empty() : bool
+ operator int() «constructor»
+ operator [ ](x : int) : int
+ operator <<(e : int) : STACK&
+ operator >>(e : int&) : STACK&
+ operator =(s : const STACK&) : STACK&
+ print()
+ ~ STACK() «destructor»
+ getmax() : const int

图 1-1 系统 UML 图

### 2.2.1 系统总体流程

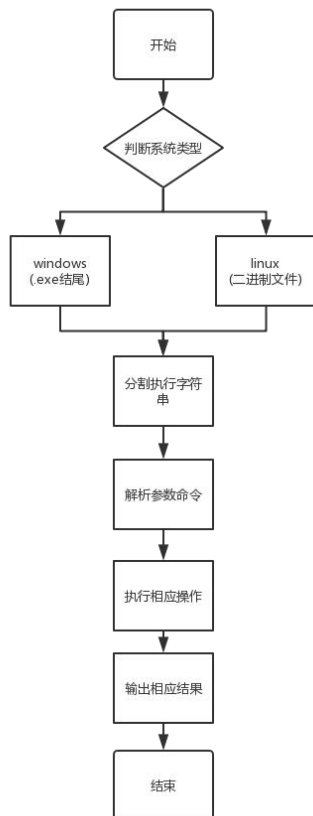


图 1-2 系统总体流程图

### 2.1.3 系统参数

表 1-1 系统参数表

命令	是否含有参数值	参数值含义
-S		设定栈队或队列大小为 a
-I i	√	入 i 个元素
-O o	√	出 o 个原色
-C		深拷贝构造
-A a	√	深拷贝赋值
-N		栈中剩余元素个数
-G g	√	表示得到栈中下标为 g 的元素

### 2.1.3 模块接口

#### 1) 操作系统判断

描述: 进行系统判断, 调用相应的函数接口

输入: 系统宏定义

输出: debug(操作系统类型)

#### 2) 调试模块

描述: 利用宏定义开关进行编译选项

输入: 程序编译类型 debug/release

输出: 是否输出 log

#### 3) 主函数

描述: 根据相应的操作系统类型调用不同的子程序

输入: 操作系统类型

调用: 模块主函数

#### 4) 模块主函数

描述: 根据相应的操作系统类型调用不同的子程序

输入: 输入参数

调用: 相应操作函数

输出: 操作结果

#### 5) Set/get 方法模块

描述: 利用宏定义封装需要 set/get

输入: 变量名称, 变量权限

输出: 变量声明/相应的 set/get 方法。

## 2.2 详细设计

### 2.2.1 数据结构类定义

```
class STACK{
    int *const elems;    //申请内存用于存放栈的元素
    const int max;       //栈能存放的最大元素个数
    PropertyBuilderByName(int,pos, private)public:
    STACK(int m);        //初始化栈: 最多存m 个元素
    STACK(const STACK&s); //用栈s 拷贝初始化栈
    virtual int size ( ) const; //返回栈的最大元素个数 max
    virtual bool full ( ) const; //返回栈的最大元素个数 max
    virtual operator int ( ) const; //返回栈的实际元素个数 pos
}
```

```

virtual int operator[ ] (int x) const;    //取下标x处的栈元素,第1个元素x=0
virtual STACK& operator<<(int e);        //将e入栈,并返回栈
virtual STACK& operator>>(int &e);        //出栈到e,并返回栈
virtual STACK& operator=(const STACK&s);  //赋s给栈,并返回被赋值的栈
virtual void print( ) const;              //打印栈
virtual ~STACK( );                        //销毁栈
const int getmax()
{
    return this->max;
};

```

### 2.2.2 数据操作设计

```

STACK(int m);                            //初始化栈:最多存m个元素
STACK(const STACK&s);                    //用栈s拷贝初始化栈
virtual int size ( ) const;               //返回栈的最大元素个数max
virtual bool full ( ) const;              //返回栈的最大元素个数max
virtual operator int ( ) const;           //返回栈的实际元素个数pos
virtual int operator[ ] (int x) const;    //取下标x处的栈元素,第1个元素x=0
virtual STACK& operator<<(int e);        //将e入栈,并返回栈
virtual STACK& operator>>(int &e);        //出栈到e,并返回栈
virtual STACK& operator=(const STACK&s);  //赋s给栈,并返回被赋值的栈
virtual void print( ) const;              //打印栈
virtual ~STACK( );                        //销毁栈
const int getmax()
{
    return this->max;
}

```

1) STACK(int m);

a) 描述:

初始化 p 指向的栈: 最多 m 个元素

b) 入口参数:

int m                      m 个元素

c) 出口参数:

无.

2) STACK(const STACK&s);

a) 描述:

用栈 s 拷贝初始化栈

b) 入口参数:

const STACK&s      用户栈 s

c) 出口参数:

无.

3) virtual int size ( ) const;

a) 描述:

返回栈的最大元素个数 max

b) 入口参数:

STACK \*const p      用户栈

c) 出口参数:

无

4) virtual int size ( ) const;

a) 描述:

返回栈的最大元素个数 max

b) 入口参数:

STACK \*const p      用户栈

c) 返回:

Int                      最大元素个数

5) virtual bool full ( ) const;

a) 描述: 返回栈是否满了

b) 入口参数:

STACK &p              用户栈引用

c) 返回:

bool

6) virtual operator int ( ) const;

a) 描述:

返回栈的实际元素个数

b) 入口参数:

STACK &p            用户栈引用

c) 返回:

int                    栈的实际元素个数

7) virtual int operator[ ] (int x) const;

a) 描述:

取下标 x 处的栈元素, 第 1 个元素 x=0

b) 入口参数:

STACK &p            用户栈引用

Int x                下标

c) 返回:

int                    下标 x 处的栈元素值

8) virtual STACK& operator<<(int e);

a) 描述:

将 e 入栈,并返回栈

b) 入口参数:

STACK &p            用户栈引用

Int e                入栈元素 e

c) 返回:

STACK& p            用户栈引用

9) virtual STACK& operator>>(int &e);

a) 描述:

出栈到 e,并返回栈

b) 入口参数:

STACK &p            用户栈引用

Int e                入栈元素 e



- c) 返回:  
STACK& p          用户栈引用

10) virtual STACK& operator=(const STACK&s);

- a) 描述:  
赋 s 给栈,并返回被赋值的栈
- b) 入口参数:  
const STACK&s      现存栈 s
- c) 返回:  
STACK& p          用户栈引用

11) virtual void print( ) const;

- a) 描述:  
打印栈
- b) 入口参数:  
STACK &p          用户栈引用
- c) 返回:  
void

12) virtual ~STACK( );

- a) 描述:  
析构函数
- b) 入口参数:  
void
- c) 返回:  
void

13) const int    getmax()

- a) 描述:  
返回 max
- b) 入口参数:  
void
- c) 返回:  
const int max      返回 private const int max

### 2.2.3 分模块设计

#### 1) 操作系统判断

描述: 进行系统判断, 调用相应的函数接口

输入: 系统宏定义

输出: debug(操作系统类型)

#### 2) 调试模块

描述: 利用宏定义开关进行编译选项

输入: 程序编译类型 debug/release

输出: 是否输出 log

#### 3) 主函数

描述: 根据相应的操作系统类型调用不同的子程序

输入: 操作系统类型

调用: 模块主函数

#### 4) 模块主函数

描述: 根据相应的操作系统类型调用不同的子程序

输入: 输入参数

调用: 相应操作函数

输出: 操作结果

#### 5) Set/get 方法模块

描述: 利用宏定义封装需要 set/get

输入: 变量名称

输出: 变量声明/相应的 set/get 方法。

```
#define PropertyBuilderByName(type, name, access_permission)\
    access_permission:\
        type name;\
    public:\
        inline void set##name(type v) {\
            name = v;\
        }\
        inline type get##name() {\
            return name;\
        }\

#define PointerPropertyBuilderByName(type, name, access_permission)\
    access_permission:\
        type* name;\
    public:\
        inline void set##name(type* v){\
            name = v;\
        }
```

```
    }\n    inline type* get##name(){\n        return name;\n    }\n\n#define constPropertyBuilderByName(type, name, access_permission)\n    access_permission:\n        const type name;\n    public:\n        inline void set##name(type v) {\n            name = v;\n        }\n        inline type get##name() {\n            return name;\n        }\n    }
```

## 3 软件开发

### 3.1 软件开发环境

Language: C++  
Text Editor: Visual Studio Code 1.29.1  
Compiler: 8.1.1 20180531 (GCC)  
Mingw-gcc 交叉编译  
Debugger: GNU gdb (GDB) 8.2  
Architecture: x64  
OS: Manjaro 18.0.0 Illyria  
Kernel: x86\_64 Linux 4.19.1-1-MANJARO

### 3.2 软件构建过程

在 Linux 下使用 x86\_64-w64-mingw32-g++ 构建, 或者在 Windows 下使用 msys2 中的 mingw 交叉编译器构建。

#### 3.2.1 编译生成

分别编译生成

lab3.o

main.o

#### 3.2.2 链接文件

链接 lab3.o 和 main.o

## 4 软件测试

### 4.1 软件测试环境

Language: C++  
Text Editor: Visual Studio Code 1.29.1  
Compiler: 8.1.1 20180531 (GCC)  
Mingw-gcc 交叉编译  
Architecture: x64  
OS: Manjaro 18.0.0 Illyria  
Windows 10  
Kernel: x86\_64 Linux 4.19.1-1-MANJARO

## 4.2 软件测试内容

### 4.2.1 测试程序



图 4-1 软件运行测试截图

### 4.2.2 测试样例

正在测试 C:/msys64/HUST\_CPP\_Labs/lab2/bin/U201614898\_2.exe.....

执行命令: C:/msys64/HUST\_CPP\_Labs/lab2/bin/U201614898\_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:/msys64/HUST\_CPP\_Labs/lab2/bin/U201614898\_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:/msys64/HUST\_CPP\_Labs/lab2/bin/U201614898\_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:/msys64/HUST\_CPP\_Labs/lab2/bin/U201614898\_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:/msys64/HUST\_CPP\_Labs/lab2/bin/U201614898\_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:/msys64/HUST\_CPP\_Labs/lab2/bin/U201614898\_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:/msys64/HUST\_CPP\_Labs/lab2/bin/U201614898\_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:/msys64/HUST\_CPP\_Labs/lab2/bin/U201614898\_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:/msys64/HUST\_CPP\_Labs/lab2/bin/U201614898\_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:/msys64/HUST\_CPP\_Labs/lab2/bin/U201614898\_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

## 5 特点与不足

### 5.1 技术特点

- 1) 使用宏开关进行 debug 调试生成 debug 版本，在正式 release 版本时关闭宏开关，不进行调试信息输出，从而达到实验要求。
- 2) 提供公共的函数接口达到整个实验系统通用，对于不同的可执行文件，执行不同的 `stack_main` 或者 `queen_main`，从而达到代码的复用。
- 3) 对于不同的操作系统进行了判断，从而达到跨平台兼容。
- 4) 代码命名规范，采用驼峰命名法，且注释规范

### 5.2 不足和改进的建议

- 1) 数据结构元素为整型，未采用模板进行实现，故栈的可复用性不高
- 2) 数据结构使用的正确性约束由程序进行

## 6 过程和体会

### 6.1 遇到的主要问题和解决方法

采用 C++ 语言进行开发，过程较为简单，主要为了比较 C 语言与 C++ 语言的差异，交叉编译遇到一些麻烦，因为某些 Linux API 在存在环境依赖，故需要将依赖打包。

### 6.2 课程设计的体会

运算符重载为 C++ 所独特的功能，且编译器实现了较好的类型转换，使运算灵活，为开发高性能计算库提供了方便。

而实际使用中可以将 `<<` 和 `>>` 进行流式重载，从而直接从输入输出进行流插入和流提取操作，从而无需中间变量，且方便进行数据流管理。



## 附录 源码和说明

### 文件清单及其功能说明

#### 文件目录结构

采用驼峰命名法，所有的类名为当前类的数据结构类型

- 1) .h 文件为头文件用于函数和数据结构的声明
- 2) .cpp 文件用于函数的实现和相关数据的实现
- 3) UTF8 编码

- bin
  - 对应的二进制文件和所依赖的库打包
- include
  - 头文件：函数和数据结构的声明
- src
  - .cpp 文件：相应的函数和数据结构实现和接口调用
- obj
  - .obj 中间文件
- debugLog
  - debugLog：程序的输出校验文件
- MakeFile
  - MakeFile 文件

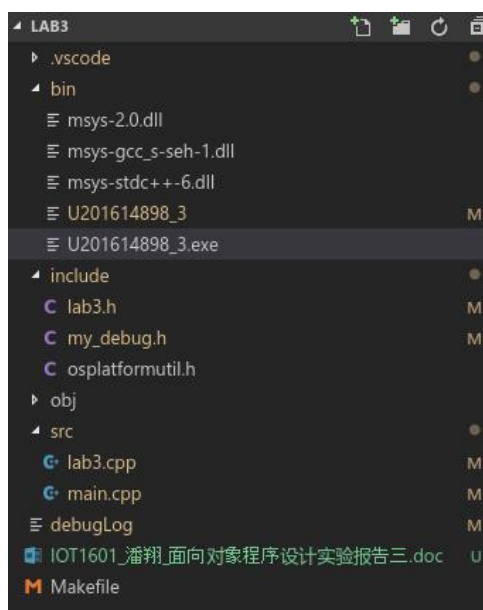


图 附录-1 文件目录结构

## 用户使用说明书

### Make

cd lab3

make

## 源代码

### Lab3.h

```
/* FileName:lab3.h
 * Author:      Hover
 * E-Mail:      hover@hust.edu.cn
 * GitHub:      HoverWings
 * Description:  The definenation of STACK
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "my_debug.h"
int stack_main(int argc, char *argv[]);

class STACK
{
    int  *const  elems; //申请内存用于存放栈的元素
    const int    max;   //栈能存放的最大元素个数
    // int    pos;      //栈实际已有元素个数，栈空时 pos=0;

    // PointerPropertyBuilderByName(int *const, elems, public)
    // PropertyBuilderByName(const int,max, public)
    // constPropertyBuilderByName(int,max,private)
    PropertyBuilderByName(int,pos,private)
public:
    STACK(int m);                //初始化栈：最多存 m 个元素
    STACK(const STACK&s);        //用栈 s 拷贝初始化栈
    virtual int  size () const;  //返回栈的最大元素个数 max
    virtual bool full () const;  //返回栈是否满了
}
```

```
virtual operator int ( ) const;           //返回栈的实际元素个数 pos
virtual int operator[ ] (int x) const; //取下标 x 处的栈元素，第 1 个元素 x=0
virtual STACK& operator<<(int e);       //将 e 入栈,并返回栈
virtual STACK& operator>>(int &e);      //出栈到 e,并返回栈
virtual STACK& operator=(const STACK&s); //赋 s 给栈,并返回被赋值的栈
virtual void print( ) const;            //打印栈
virtual ~STACK( );                      //销毁栈
const int  getmax()
{
    return this->max;
}
};
```

### my\_debug.h

```
/* FileName:    my_debug.h
 * Author:      Hover
 * E-Mail:      hover@hust.edu.cn
 * GitHub:      HoverWings
 * Description: The debug macro and the file redirect macro
 * Set/Get operation
 */
```

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

```
using namespace std;
// FILE *fd;//文件指针
// strcat(fname, ".txt");
// fd=fopen(fname, "w+");
```

```
#define ECHO_COLOR_NONE          "\033[0;0m"
#define ECHO_COLOR_GREEN         "\033[0;32m"
```

```
#define __DEBUG
#undef __DEBUG
```

```
#ifdef __DEBUG
#define debug(fmt,args...)    \
    printf(ECHO_COLOR_GREEN "\nDebug: " fmt "\n" ECHO_COLOR_NONE, ##args);
// printf(ECHO_COLOR_GREEN "Debug: " fmt "(file: %s, func: %s, line: %d)\n"
ECHO_COLOR_NONE, ##args, __FILE__, __func__, __LINE__);
    // printf(ECHO_COLOR_GREEN fmt ECHO_COLOR_NONE);
    // printf(ECHO_COLOR_GREEN "Debug: " fmt "\n");

#else
#define debug(fmt, args...)
#endif

#ifdef __DEBUG
#define cdebug(fmt)  cout <<  fmt;
#else
#define cdebug(fmt)
#endif

// PropertyBuilderByName 用于生成类的成员变量
// 并生成 set 和 get 方法
// type 为变量类型
// access_permission 为变量的访问权限(public, priavte, protected)

#define PropertyBuilderByName(type, name, access_permission)\
    access_permission:\
        type name;\
    public:\
    inline void set###name(type v) {\
        name = v;\
    }\
    inline type get###name() {\
        return name;\
    }
```

```
#define PointerPropertyBuilderByName(type, name, access_permission)\
    access_permission:\
        type* name;\
public:\
    inline void set##name(type* v){\
        name = v;\
    }\
    inline type* get##name(){\
        return name;\
    }
```

```
#define constPropertyBuilderByName(type, name, access_permission)\
    access_permission:\
        const type name;\
public:\
    inline void set##name(type v) {\
        name = v;\
    }\
    inline type get##name() {\
        return name;\
    }
```

### osplatformutil.h

```
/* FileName:osplatformutil.h
```

```
 * Author:      Hover
```

```
 * E-Mail:      hover@hust.edu.cn
```

```
 * GitHub:      HoverWings
```

```
 * Description:  osplatformutil marco judgement
```

```
 */
```

```
#ifndef OSPLATFORMUTIL_H
```

```
#define OSPLATFORMUTIL_H
```

```
/*
```

The operating system, must be one of: (I\_OS\_x)

DARWIN - Any Darwin system (macOS, iOS, watchOS, tvOS)

ANDROID - Android platform  
WIN32 - Win32 (Windows 2000/XP/Vista/7 and Windows Server 2003/2008)  
WINRT - WinRT (Windows Runtime)  
CYGWIN - Cygwin  
LINUX - Linux  
FREEBSD - FreeBSD  
OPENBSD - OpenBSD  
SOLARIS - Sun Solaris  
AIX - AIX  
UNIX - Any UNIX BSD/SYSV system

\*/

```
#define OS_PLATFORM_UTIL_VERSION 1.0.0.180723
```

```
// DARWIN
```

```
#if defined(__APPLE__) && (defined(__GNUC__) || defined(__xlc__) || defined(__xlc__))
```

```
# include <TargetConditionals.h>
```

```
# if defined(TARGET_OS_MAC) && TARGET_OS_MAC
```

```
#   define I_OS_DARWIN
```

```
#   ifdef __LP64__
```

```
#       define I_OS_DARWIN64
```

```
#   else
```

```
#       define I_OS_DARWIN32
```

```
#   endif
```

```
# else
```

```
#   error "not support this Apple platform"
```

```
# endif
```

```
// ANDROID
```

```
#elif defined(__ANDROID__) || defined(ANDROID)
```

```
#   define I_OS_ANDROID
```

```
#   define I_OS_LINUX
```

```
// Windows
```

```
#elif !defined(SAG_COM) && (!defined(WINAPI_FAMILY) ||
```

```
WINAPI_FAMILY==WINAPI_FAMILY_DESKTOP_APP) && (defined(WIN64) ||
```

```
defined(_WIN64) || defined(__WIN64__))
```

```
#   define I_OS_WIN32
```

```
# define I_OS_WIN64
#elif !defined(SAG_COM) && (defined(WIN32) || defined(_WIN32) || defined(__WIN32__) ||
defined(__NT__))
# if defined(WINAPI_FAMILY)
#   ifndef WINAPI_FAMILY_PC_APP
#       define WINAPI_FAMILY_PC_APP WINAPI_FAMILY_APP
#   endif
#   if defined(WINAPI_FAMILY_PHONE_APP) &&
WINAPI_FAMILY==WINAPI_FAMILY_PHONE_APP
#       define I_OS_WINRT
#   elif WINAPI_FAMILY==WINAPI_FAMILY_PC_APP
#       define I_OS_WINRT
#   else
#       define I_OS_WIN32
#   endif
# else
#       define I_OS_WIN32
#   endif
//CYGWIN
#elif defined(__CYGWIN__)
# define I_OS_CYGWIN
// sun os
#elif defined(__sun) || defined(sun)
# define I_OS_SOLARIS
// LINUX
#elif defined(__linux__) || defined(__linux)
# define I_OS_LINUX
// FREEBSD
#elif defined(__FreeBSD__) || defined(__DragonFly__) || defined(__FreeBSD_kernel__)
#   ifndef __FreeBSD_kernel__
#       define I_OS_FREEBSD
#   endif
#   define I_OS_FREEBSD_KERNEL
// OPENBSD
#elif defined(__OpenBSD__)
# define I_OS_OPENBSD
```

```
// IBM AIX
#elif defined(_AIX)
#   define I_OS_AIX
#else
#   error "not support this OS"
#endif

#if defined(I_OS_WIN32) || defined(I_OS_WIN64) || defined(I_OS_WINRT)
#   define I_OS_WIN
#endif

#if defined(I_OS_WIN)
#   undef I_OS_UNIX
#elif !defined(I_OS_UNIX)
#   define I_OS_UNIX
#endif

#ifdef I_OS_DARWIN
#define I_OS_MAC
#endif
#ifdef I_OS_DARWIN32
#define I_OS_MAC32
#endif
#ifdef I_OS_DARWIN64
#define I_OS_MAC64
#endif

#endif // OSPLATFORMUTIL_H
```

### Lab3.cpp

```
/* FileName:lab3.cpp
 * Author:      Hover
 * E-Mail:      hover@hust.edu.cn
 * GitHub:      HoverWings
 * Description:  The implementation of STACK
 */
```



```

#include "lab3.h"
#include <iostream>
#include <stdlib.h>
#include <cctype>
/*
“设定栈队或队列大小-S”
“入-I”、
“出-O”、
“深拷贝构造-C”、
“深拷贝赋值-A”、
“栈中剩余元素个数-N”
*/
using namespace std;

int stack_main(int argc, char *argv[])
{
    int num;//元素个数&&入栈数字
    int out;//接受出栈元素
    STACK *s;
    STACK *p;
    int ch;
    bool fail=false;
    while ((ch = getopt(argc, argv, "S:I:O:CA:NG:")) != -1)
    {
        if(fail)
        {
            cdebug("false");
            break;
        }
        fail=false;
        debug("optind: %d\n", optind);
        switch (ch)
        {
            case 'S':
                debug("HAVE option: -S");
                debug("The argument of -S is %s", optarg);

```

```

num=atoi(optarg);
debug("%d",num);
printf("S   %d", num);
s = new STACK(num);
break;
case 'I':
    debug("HAVE option: -I");
    debug("The argument of -I is %s", optarg);
    num=atoi(optarg);
    debug("%d",num);
    if(s->getpos()==s->getmax())
    {
        fail=true;
        printf("   I");
        printf("   E");
        break;
    }
    (*s)<<num;
    // debug(argv[optind][0]);
    while(isdigit(argv[optind][0]))
    {
        num=atoi(argv[optind]);
        debug("%d",num);
        if(s->getpos()==s->getmax())
        {
            fail=true;
            printf("   I");
            printf("   E");
            break;
        }
        (*s)<<num;
        optind++;
        if(optind==argc)
        {
            break;
        }
    }

```

```

    }
    if(fail==false)
    {
        printf("  I");
        s->print();
    }
    break;
case 'O':
    debug("HAVE option: -O");
    debug("The argument of -O is %s", optarg);
    num=atoi(optarg);
    debug("%d",num);
    for (int j = 0; j < num; j++)
    {
        if (int(*s)== 0)
        {
            printf("  O  E");
            exit(0);
        }
        (*s)>>out;
    }
    printf("  O");
    s->print();
    break;
case 'C':
    debug("HAVE option: -C");
    printf("  C");
    p = s;
    s= p;
    s->print();
    break;
case 'A':
    debug("HAVE option: -A");
    debug("The argument of -A is %s", optarg);
    num=atoi(optarg);
    printf("  A");

```

```

        p = new STACK(num);
        (*p)=*s;          //assign p to s
        s = p;
        s->print();        //print current stack
        break;
    case 'N':
        debug("HAVE option: -N");
        printf("  N");
        printf("  %d", int(*s));
        break;
    case 'G':
        debug("HAVE option: -G");
        debug("The argument of -G is %s", optarg);
        num=atoi(optarg);
        printf("  G");
        if(num>s->getpos())
        {
            printf("  E");
            break;
        }
        printf("  %d", (*s)[num]);
        break;
    default:
        debug("Unknown option: %c",(char)optopt);
        break;
    }

}

return 0;
}

```

//Impletation Stack Fun

//Overload

STACK::STACK(int m): elems(m > 0 ? new int[m] : new int[0]), max(m > 0 ? m : 0)

```
{
    this->pos = 0;
    for (int i = 0; i < this->size(); i++)
    {
        this->elems[i] = 0;
    }
}
```

STACK::STACK(const STACK &s): elems(s.max > 0 ? new int[s.max] : new int[0]), max(s.max > 0 ? s.max : 0)

```
{
    this->pos = 0;
    for (int i = 0; i < (int)s && i < this->size(); i++)
    {
        (*this)<<s[i];
    }
}
```

int STACK::size() const

```
{
    return this->max;
}
```

bool STACK::full() const

```
{
    return (this->max==(int)(this->pos));
}
```

STACK::operator int(void) const

```
{
    return (int)(this->pos);
}
```

int STACK::operator[](int x) const

```
{
    // out of range check
    if (x < 0 || x >= (int)(*this)) return 0;
    return this->elems[x];
}
```

STACK& STACK::operator<<(int e)

```
{
    // full check
    // if (this->size() <= (int)(*this)) return *this;
    cdebug(this->elems[this->pos]);
    this->elems[this->pos++] = e;
    return *this;
}
```

STACK& STACK::operator>>(int &e)

```
{
    // empty check
    if ((int)(*this) <= 0)
    {
        e = 0;
        return *this;
    }

    e = this->elems[--this->pos];
    return *this;
}
```

STACK& STACK::operator=(const STACK &s)

```
{
    this->~STACK();
    new (this) STACK(s);
    return *this;
}
```

```
void STACK::print(void) const
{
    for (int i = 0; i < (int)(*this); i++)
    {
        cout<<"    "<<(*this)[i];
    }
}
```

```
STACK::~~STACK(void)
{
    delete this->elems;
    this->pos = 0;
}
```

### **main.cpp**

```
/* FileName:main.cpp
 * Author:      Hover
 * E-Mail:      hover@hust.edu.cn
 * GitHub:      HoverWings
 * Description:  The main of program and call the interface funciton math the OS
 */

#include <iostream>
#include <unistd.h>
#include <string.h>
#include "my_debug.h"
#include "lab3.h"
#include "osplatformutil.h"

using namespace std;

//Helper Function
const char *find_file_name(const char *name)
```

```
{
    char *name_start = NULL;
    int sep = '/';
    if (NULL == name)
    {
        printf("the path name is NULL\n");
        return NULL;
    }
    name_start = (char *)strchr(name, sep);
    return (NULL == name_start)?name:(name_start + 1);
}
```

//Judge Funciton

```
int main(int argc, char *argv[])
{
    #if defined I_OS_LINUX
    debug("this is linux");
    // cout<<argv[0];
    const char* file_name;
    file_name=find_file_name(argv[0]);
    if(strcmp(file_name, "U201614898_1") == 0)
    {
        debug("stack!");
        stack_main(argc,argv);
    }
    else if(strcmp(file_name, "U201614898_2") == 0)
    {
        debug("stack!");
        stack_main(argc,argv);
    }
    else if(strcmp(file_name, "U201614898_3") == 0)
    {
        debug("queue!");
        stack_main(argc,argv);
    }
}
```



```

    }
    else if(strcmp(file_name, "U201614898_4") == 0)
    {
        debug("queue!");
        // queue_main(argc,argv);
    }
    #elif defined I_OS_WIN32
    cout<<"this is windows"<<endl;
    #elif defined I_OS_CYGWIN
    debug("this is cygwin");
    const char* file_name;
    file_name=find_file_name(argv[0]);
    if(strcmp(file_name, "U201614898_1") == 0)
    {
        debug("stack!");
        stack_main(argc,argv);
    }
    else if(strcmp(file_name, "U201614898_2") == 0)
    {
        debug("stack!");
        stack_main(argc,argv);
    }
    else if(strcmp(file_name, "U201614898_3") == 0)
    {
        debug("queue!");
        stack_main(argc,argv);
    }
    else if(strcmp(file_name, "U201614898_4") == 0)
    {
        debug("queue!");
        // queue_main(argc,argv);
    }
    #endif
    return 0;
}

```

## MakeFile

```
SOURCES = $(shell find . -path ./test -prune -o -name "*.cpp" -print)
OBJECTS = $(patsubst %.cpp, %.o, $(SOURCES))
C_OBJ = $(patsubst %.o, $(OBJ_DIR)/%.o, $(notdir $(OBJECTS)))
PROG=U201614898_3
```

```
# macro for tools
# CC = x86_64-w64-mingw32-g++
CC = g++
RM = rm -fr
MV = mv
CP = cp -fr
MKDIR = mkdir -p
# BROWSER = google-chrome
```

```
# macro for flags
FLAGS = -c -Wall -g $(addprefix -I, $(INCLUDE))
```

```
# path macro
BIN_DIR = ./bin
OBJ_DIR = ./obj
```

```
# include macro
INCLUDE += ./include/
```

```
all: $(OBJECTS) link
```

```
.cpp.o:
    @echo Compiling C Source Files $< ...
    $(MKDIR) $(OBJ_DIR)
    $(CC) $(FLAGS) $< -o $@
    $(MV) $@ $(OBJ_DIR)/$(notdir $@)
```

```
link:
```

@echo Linking Binary File

\$(MKDIR) \$(BIN\_DIR)

\$(CC) \$(C\_OBJ) -o \$(BIN\_DIR)/\$(PROG)

@echo

@echo Build Success!

.PHONY:run

run:

./bin/\$(PROG)

