

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：面向过程的整型栈编程

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。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。

```
struct STACK
{
    int *elems;    //申请内存用于存放栈的元素
    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;
};

void initSTACK(STACK *const p, int m);           //初始化 p 指向的栈：最多 m 个元素
void initSTACK(STACK *const p, const STACK&s);    //用栈 s 初始化 p 指向的栈
int  size (const STACK *const p);                //返回 p 指向的栈的最大元素个数 max
int  howMany (const STACK *const p);             //返回 p 指向的栈的实际元素个数 pos
int  getelem (const STACK *const p, int x);       //取下标 x 处的栈元素
STACK *const push(STACK *const p, int e);         //将 e 入栈，并返回 p
STACK *const pop(STACK *const p, int &e);         //出栈到 e，并返回 p
STACK *const assign(STACK*const p, const STACK&s); //赋 s 给 p 指的栈,并返回 p
void print(const STACK*const p);                 //打印 p 指向的栈
void destroySTACK(STACK*const p);                //销毁 p 指向的栈
```

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，main 调用模块，数据结构模块，调试模块。

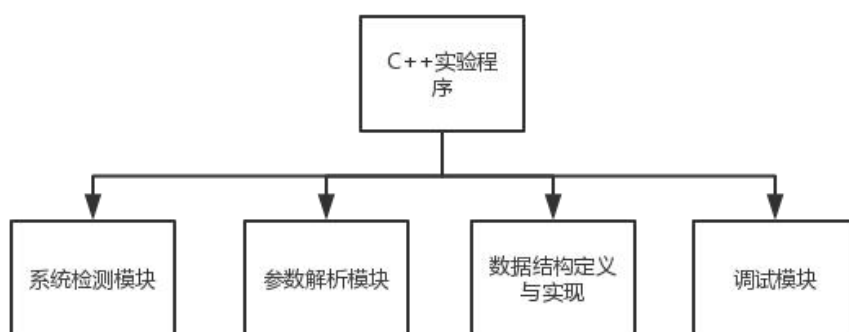


图 1-1 系统整体架构图

2.2.1 系统总体流程

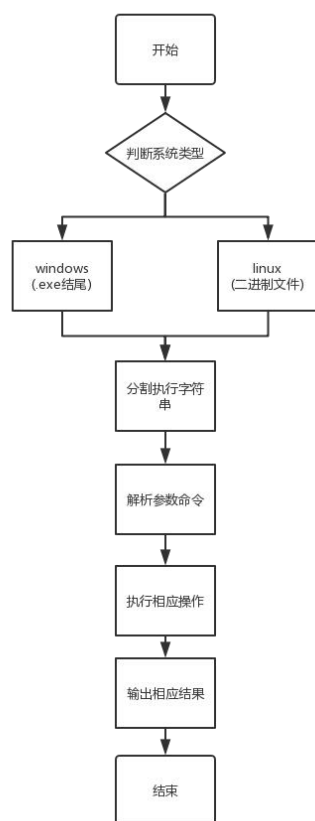


图 1-2 系统总体流程图

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

2.2 详细设计

2.2.1 数据结构定义

```
typedef struct STACK
{
    int *elems;           //申请内存用于存放栈的元素
    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
}STACK;
```

2.2.2 数据操作设计

```
// Stack Function
void initSTACK(STACK *const p, int m);           //初始化 p 指向的栈: 最多 m 个元素
void initSTACK(STACK *const p, const STACK&s);   //用栈 s 初始化 p 指向的栈
int size (const STACK *const p);                //返回 p 指向的栈的最大元素个数 max
int howMany (const STACK *const p);              //返回 p 指向的栈的实际元素个数 pos
int getelem (const STACK *const p, int x);       //取下标 x 处的栈元素
STACK *const push(STACK *const p, int e);        //将 e 入栈, 并返回 p
STACK *const pop(STACK *const p, int &e);        //出栈到 e, 并返回 p
```

```
STACK *const assign(STACK*const p, const STACK&s); //赋 s 给 p 指的栈, 并返回 p
void print(const STACK*const p);                //打印 p 指向的栈
void destroySTACK(STACK*const p);                //销毁 p 指向的栈
```

1) 初始化栈

```
void initSTACK(STACK *const p, int m);
```

- a) 入口参数:
 - i. STACK *const p 用户栈
 - ii. int m m 个元素
- b) 出口参数: 无.
- c) 描述: 初始化 p 指向的栈: 最多 m 个元素

```
void initSTACK(STACK *const p, const STACK&s);
```

- a) 入口参数:
 - i. STACK *const p 用户栈
 - ii. const STACK&s 现存栈
- b) 出口参数: 无
- c) 描述: 用栈 s 初始化 p 指向的栈

2) 输出栈大小

```
int size(const STACK *const p);
```

- a) 入口参数:
 - const STACK *const p 用户栈
- b) 出口参数: int size 栈大小(容量)
- c) 描述: 返回 p 指向的栈的最大元素个数 max

3) 输出栈容量

```
int howMany(const STACK *const p);
```

- a) 入口参数:
 - const STACK *const p 用户栈
- b) 出口参数: int nowSize 栈当前大小
- c) 描述: 返回 p 指向的栈的实际元素个数 pos

4) 输出栈元素

int getelem (const STACK *const p, int x);

- a) 入口参数:
 - i. STACK *const p 用户栈
 - ii. Int x 下标 x
- b) 出口参数: int elem 下标 x 处的栈元素
- c) 描述: 取下标 x 处的栈元素

5) 入栈

STACK *const push(STACK *const p, int e);

- a) 入口参数:
 - i. STACK *const p 用户栈 p
 - ii. int &e e 的引用
- b) 出口参数: STACK *const 栈当前大小
- c) 描述: 将 e 入栈, 并返回 p

6) 出栈

STACK *const pop(STACK *const p, int &e);

- a) 入口参数:
 - i. STACK *const p 用户栈 p
 - ii. int &e e 的引用
- b) 出口参数: STACK *const p 用户栈 p 的引用
- c) 描述: 将 e 入栈, 并返回 p

7) 栈赋值

STACK *const assign(STACK*const p, const STACK&s);

- a) 入口参数:
 - i. STACK *const p 用户栈 p
 - ii. const STACK&s 当前栈 s 的引用
- b) 出口参数: STACK *const p 用户栈 p 的引用
- c) 描述: 赋 s 给 p 指的栈,并返回 p

8) 栈打印

void print(const STACK*const p);

- a) 入口参数:
 - STACK *const p 用户栈 p
- b) 出口参数: 无

c) 描述: 打印 p 指向的栈

9) 栈销毁

`void destroySTACK(STACK*const p);`

a) 入口参数:

STACK *const p 用户栈 p

b) 出口参数: 无

c) 描述: 销毁 p 指向的栈

2.2.3 分模块设计

1) 操作系统检测

检测当前操作系统, 并进行相应的底层程序调用

2) 参数解析模块

解析参数

3) 用户输入检查

进行对应操作的用户输入检查

4) 数据结构管理

对于相应的操作指令调用下层数据结构

5) 数据结构实现

6) 调试模块

在调试过程中, 通过宏开关控制中间 log 的输出

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

lab1.o

main.o

3.2.2 链接文件

链接 lab1.o 和 main.o

4 软件测试

对照题目要求, 构造测试例, 给出程序界面截图, 举证题目要求的功能 (以及自行补充的功能) 已实现。

分析测试效果。

注意: 已实现但未在报告中主动举证的功能可能被当作没有实现。

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/lab1/bin/U201614898_1.exe.....

执行命令: C:/msys64/HUST_CPP_Labs/lab1/bin/U201614898_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:/msys64/HUST_CPP_Labs/lab1/bin/U201614898_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:/msys64/HUST_CPP_Labs/lab1/bin/U201614898_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:/msys64/HUST_CPP_Labs/lab1/bin/U201614898_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:/msys64/HUST_CPP_Labs/lab1/bin/U201614898_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:/msys64/HUST_CPP_Labs/lab1/bin/U201614898_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:/msys64/HUST_CPP_Labs/lab1/bin/U201614898_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:/msys64/HUST_CPP_Labs/lab1/bin/U201614898_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:/msys64/HUST_CPP_Labs/lab1/bin/U201614898_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:/msys64/HUST_CPP_Labs/lab1/bin/U201614898_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

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。

5.2 不足和改进的建议

- 1) 数据结构元素为整形, 未采用模板进行实现, 故栈的可复用性不高

- 2) 使用 C 语言进行开发，未实现良好封装

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 存在环境依赖，故需要将依赖打包。

6.2 课程设计的体会

本次实验较为简单，使用 C 语言进行数据结构的实现，在调用处进行边界条件的判定，增强软件的鲁棒性。

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

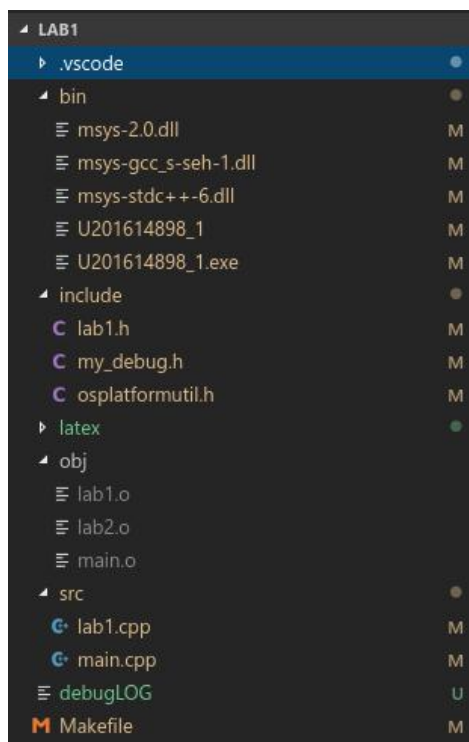


图 附录-1 文件目录结构

用户使用说明书

Make

```
cd lab1
```

```
make
```

cmake

```
cd lab1
```

```
mkdir build
```

```
Cmake ..
```

源代码

lab1.h

```
/* FileName:lab1.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>

/// @brief the stack structure
typedef struct STACK
{
    int *elems; //申请内存用于存放栈的元素
    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
}STACK;

// Helper Function
int stack_main(int argc, char *argv[]);

// Stack Function
/// @brief initialize stack with size
/// @param[in] p the stack pointer
/// @param[in] m the size of the stack
void initSTACK(STACK *const p, int m);           //初始化 p 指向的栈：最多 m 个元素

/// @brief initialize a stack with another stack
/// @param[in] p the stack pointer to initialize
/// @param[in] s the stack to copy
void initSTACK(STACK *const p, const STACK&s);    //用栈 s 初始化 p 指向的栈

/// @brief return the capacity of the stack
/// @param[in] p the stack pointer
/// @return the capacity of the stack
int size(const STACK *const p);                  //返回 p 指向的栈的最大元素个数 max

/// @brief return the element number of the stack
/// @param[in] p the stack pointer
/// @return the element number of the stack
int howMany(const STACK *const p);               //返回 p 指向的栈的实际元素个数 pos

```

```

/// @brief get the element from the stack
/// @param[in] p the stack pointer
/// @param[in] x the index of the element
/// @return the element
int getelem (const STACK *const p, int x);           //取下标 x 处的栈元素

/// @brief push an element to a stack
/// @param[in] p the the stack pointer
/// @param[in] e the element to push
/// @return the stack pointer
STACK *const push(STACK *const p, int e);           //将 e 入栈，并返回 p

/// @brief pop an element from a stack
/// @param[in] p the stack pointer
/// @param[out] e the element to pop
/// @return the stack pointer
STACK *const pop(STACK *const p, int &e);           //出栈到 e，并返回 p

/// @brief assign the elements in s to elements in p
/// @param[in] p the stack to assign
/// @param[in] s the stack to copy from
/// @return the stack after assigned
STACK *const assign(STACK*const p, const STACK&s); //赋 s 给 p 指的栈,并返回 p

/// @brief print elements in stack
/// @param[in] p the stack pointer
void print(const STACK*const p);                     //打印 p 指向的栈

/// @brief de-construct the stack
/// @param[in] p the stack pointer
void destroySTACK(STACK*const p);                   //销毁 p 指向的栈

```

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
*/

#include <stdio.h>

#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

#define __FILE
// #undef __FILE

#ifdef __FILE
#define my_printf(fmt,args...)  printf(fmt, ##args);
#else
#define my_printf(fmt)  fprintf(fd, fmt);
#endif
```

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
```

lab1.cpp

```
/* FileName:lab1.cpp
 * Author:      Hover
 * E-Mail:      hover@hust.edu.cn
 * GitHub:      HoverWings
 * Description:  The implementation of STACK
 */

#include "lab1.h"
#include "my_debug.h"
#include <stdlib.h>
#include <ctype.h>
/*
“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”
*/

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 *p = (STACK *)malloc(sizeof(STACK));
    STACK *s = (STACK *)malloc(sizeof(STACK));
    STACK *ss;
    int ch;
    bool fail=false;
```

```

while ((ch = getopt(argc, argv, "S:I:O:CA:NG:")) != -1)
{
    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);
            initSTACK(p, num);
            break;
        case 'I':
            debug("HAVE option: -I");
            debug("The argument of -I is %s", optarg);
            num=atoi(optarg);
            debug("%d",num);
            if(p->pos==p->max)
            {
                fail=true;
                printf("   I");
                printf("   E");
                break;
            }
            p = push(p, num);
            // debug(argv[optind][0]);
            while(isdigit(argv[optind][0]))
            {
                num=atoi(argv[optind]);
                debug("%d",num);
                if(p->pos==p->max)
                {
                    fail=true;
                    printf("   I");

```

```

        printf("  E");
        break;
    }
    p = push(p, num);
    optind++;
    if(optind==argc)
    {
        break;
    }
}
if(fail==false)
{
    printf("  I");
    print(p);
}
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 (p->pos == 0)
        {
            printf("  O  E");
            exit(0);
        }
        p = pop(p, out);
    }
    printf("  O");
    print(p);
    break;
case 'C':
    debug("HAVE option: -C");
    printf("  C");

```

```

ss = (STACK *)malloc(sizeof(STACK));
initSTACK(ss, *p);
destroySTACK(p);
p = ss;
print(p);
break;
case 'A':
    debug("HAVE option: -A");
    debug("The argument of -A is %s", optarg);
    num=atoi(optarg);
    printf("  A");
    initSTACK(s, num);
    s = assign(s, *p); //将栈 p 赋值给新栈 s
    destroySTACK(p); //销毁旧栈
    p = s;
    print(p);          //打印当前栈
    break;
case 'N':
    debug("HAVE option: -N");
    printf("  N");
    printf("  %d", howMany(p));
    break;
case 'G':
    debug("HAVE option: -G");
    debug("The argument of -G is %s", optarg);
    num=atoi(optarg);
    printf("  G");
    if(num>p->pos)
    {
        printf("  E");
        break;
    }
    printf("  %d", getelem(p,num));
    break;
default:
    debug("Unknown option: %c",(char)optopt);

```



```

        break;
    }
}

//Impletation Stack Fun

//Overload
//D:初始化 p 指向的栈： 最多 m 个元素
void initSTACK(STACK *const p, int m)
{
    p->elems = (int *)malloc(m * sizeof(int));
    p->max = m;
    p->pos = 0;
}

//D:用栈 s 初始化 p 指向的栈
void initSTACK(STACK *const p, const STACK &s)
{
    p->elems = (int *)malloc(s.max * sizeof(int));
    for (int j = 0; j < s.pos; j++)
        p->elems[j] = s.elems[j];
    p->max = s.max;
    p->pos = s.pos;
}

//D:返回 p 指向的栈的最大元素个数 max
int size(const STACK *const p)
{
    return p->max;
}

//D:返回 p 指向的栈的实际元素个数 pos
int howMany(const STACK *const p)
{
    return p->pos;
}

```

//D:取下标 x 处的栈元素

```
int getelem(const STACK *const p, int x)
{
    return p->elems[x];
}
```

//D:将 e 入栈，并返回 p

```
STACK *const push(STACK *const p, int e)
{
    p->elems[(p->pos)++] = e;
    return p;
}
```

//D:出栈到 e，并返回 p

```
STACK *const pop(STACK *const p, int &e)
{
    e = p->elems[--(p->pos)];
    return p;
}
```

//D:赋 s 给 p 指的栈，并返回 p

```
STACK *const assign(STACK *const p, const STACK &s)
{
    free(p->elems);
    p->elems = (int *)malloc(s.max * sizeof(int));
    for (int j = 0; j < s.pos; j++)
        p->elems[j] = s.elems[j];
    p->max = s.max;
    p->pos = s.pos;
    return p;
}
```

//D:打印 p 指向的栈

```
void print(const STACK *const p)
{

```

```

    for (int j = 0; j < p->pos; j++)
    {
        printf("  %d", p->elems[j]);
    }
}
。
//D:销毁 p 指向的栈
void destroySTACK(STACK *const p)
{
    free(p->elems);
    free(p);
}

```

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 "lab1.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 Function

```

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!");
        // queue_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!");
        // queue_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_1

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}