2021 高级程序设计 作业一

191300051

F. Y. Wong

从数据与过程的角度，简述抽象与封装的区别

对于一个程序实体而言，抽象是该程序实体外部可观察到的行为，不考虑外部是如何实现的。封装是指把该程序实体内部的具体实现细节对使用者隐藏起来，只对外提供一个接口。

从数据来看

数据抽象
只描述对数据能实施哪些操作以及这些操作之间的关系，数据的使用者不需要知道数据的具体表示形式。

数据封装
把数据及其操作作为一个整体来进行实现，其中，数据的具体表示被隐藏起来（使用者不可见，或不可直接访问），对数据的访问（使用）只能通过提供的操作（对外接口）来完成。

从过程来看

过程抽象
用一个名字来代表一段完成一定功能的程序代码，代码的使用者只需要知道代码的名字以及相应的功能，而不需要知道对应的程序代码是如何实现的。

过程封装
把命名代码的具体实现隐藏起来（对使用者不可见，或不可直接访问），使用者只能通过代码名字来使用相应的代码。
命名代码所需要的数据是通过参数（或全局变量）来获得，计算结果通过返回机制（或全局变量）返回。

简述面向对象与面向过程的程序设计的区别；列举两个更加适合面向对象的场景并说明理由

面向过程

以功能为中心，强调过程（功能）抽象，但数据与操作分离，二者联系松散。
实现了操作的封装，但数据是公开的，数据缺乏保护。
按子程序划分模块，模块边界模糊。
子程序往往针对某个程序而设计，这使得程序难以复用。
功能易变，程序维护困难。
基于子程序的解题方式与问题空间缺乏对应。

面向对象

以数据为中心，强调数据抽象，操作依附于数据，二者联系紧密。
实现了数据的封装，加强了数据的保护。
按对象类划分模块，模块边界清晰。
对象类往往具有通用性，再加上继承机制，使得程序容易复用。
对象类相对稳定，有利于程序维护。
基于对象交互的解题方式与问题空间有很好的对应。

更加适合面向对象的场景

大型的项目：以数据为中心，使操作依附于数据，加强了操作和数据的联系。而在面向过程式的程序设计中，随着项目的不断扩大，很有可能发生不相匹配的数据和操作的使用。

代码需要大量灵活的复用的项目：利用类的继承机制，能够极大的提高代码的复用率。而面向过程式则需要重复定义多个函数。

仿照课堂所讲栈类Stack的实现,利用链表和数组分别实现队列类Queue

全部的源文件代码-可直接运行。

简短说明

- 打开Queue.cpp中的ARRAY的宏定义，调用数组实现
- 注释ARRAY宏定义，打开PTR宏定义，调用指针实现

```
//Queue.h
#ifndef __Queue_H__
#define __Queue_H__

#define RESET   "\033[0m"
#define BLACK   "\033[30m"      /* Black */
#define RED     "\033[31m"      /* Red */
#define GREEN   "\033[32m"      /* Green */
#define YELLOW  "\033[33m"      /* Yellow */
#define BLUE    "\033[34m"      /* Blue */
#define MAGENTA "\033[35m"      /* Magenta */
#define CYAN    "\033[36m"      /* Cyan */
#define WHITE   "\033[37m"      /* White */
#define BOLDBLACK   "\033[1m\033[30m"      /* Bold Black */
#define BOLDRED     "\033[1m\033[31m"      /* Bold Red */
#define BOLDGREEN   "\033[1m\033[32m"      /* Bold Green */
#define BOLDYELLOW  "\033[1m\033[33m"      /* Bold Yellow */
#define BOLDBLUE    "\033[1m\033[34m"      /* Bold Blue */
#define BOLDMAGENTA "\033[1m\033[35m"      /* Bold Magenta */
#define BOLDCYAN    "\033[1m\033[36m"      /* Bold Cyan */
#define BOLDWHITE   "\033[1m\033[37m"      /* Bold White */



#define MAXSIZE 100
#define SIZENUM 5
```

```
#endif
```

```
//Queue.cpp
#include <iostream>
#include "Queue.h"
#include <string.h>
#include <assert.h>
using namespace std;

static const int SIZES[SIZENUM] = {10, 20, 30, 50, 100};
//static const int SIZES[SIZENUM] = {1, 2, 3, 5, 10};//for test

#define ARRAY
// #define PTR

#if defined ARRAY
class Queue
{
private:
    int *stack = (int *)malloc(sizeof(int) * SIZES[0]);
    int cnt;
    int isize;

public:
    Queue();
    void enQueue(int i);
    void deQueue(int &i);
    void printAll();
};

Queue::Queue()
{
    cnt = 0;
    isize = 0;
}

void Queue::enQueue(int data)
{
    if (cnt >= SIZES[isize])
    {
        if (isize + 1 >= SIZENUM)
        {
            cout << BOLDRED << "WRONG:--- queue --- overflow\n";
            exit(-1);
            return;
        }
        else
        {
            cout << BOLDYELLOW << "WARNING:---queue---expanding\n";
            int *tmp = (int *)malloc(sizeof(int) * SIZES[isize + 1]);
            memcpy((void *)tmp, (void *)stack, SIZES[isize] * sizeof(int));
            free(stack);
            stack = tmp;
```

```
            stack[cnt] = data;
            cnt++;
            isize++;
        }
    }
    else
    {
        stack[cnt] = data;
        cnt++;
    }
//    cout << BOLDGREEN << "ADD SUCCESSFULLY\n";
}

void Queue::deQueue(int &data)
{
    if (cnt == 0)
    {
        cout << BOLDRED << "WRONG:---queue---empty\n";
        return;
    }
    cnt--;
    data = stack[0];

    for (int i=0;i<cnt;i++)
    {
        stack[i]=stack[i+1];
    }

    if (cnt <= SIZES[isize - 1])
    {
        cout << BOLDYELLOW << "WARNING:---queue---shrinking\n";
        int *tmp = (int *)malloc(sizeof(int) * SIZES[isize - 1]);
        memcpy((void *)tmp, (void *)stack, SIZES[isize - 1] * sizeof(int));
        free(stack);
        stack = tmp;
        isize--;
    }
}

void Queue::printAll()
{
    cout << BOLDBLUE << "PRINT ALL ELEMENTS\n";
    for (int i = 0; i < cnt; i++)
    {
        cout << "stack[" << i << "]=" << stack[i] << "\n";
    }
}
#elif defined PTR

class Queue
{
private:
    struct Node
    {
        int content;
        Node *next;
    } * head, *tail;
    int cnt;
```

```
public:
    Queue();
    void enQueue(int i);
    void deQueue(int &i);
    void printAll();

};

Queue::Queue()
{
    head=tail=NULL;
    cnt=0;
}

void Queue::enQueue(int i)
{
    if (cnt==0)
    {
        assert(head==NULL);
        assert(tail==NULL);
        cnt++;
        tail=head=new Node;
        tail->content=head->content=i;
        tail->next=head->next=NULL;
//        cout<<BOLDGREEN<<"PUSH SUCCESSFULLY\n";
    }
    else if (cnt>=MAXSIZE)
    {
        cout<<BOLDRED<<"WRONG:---queue---overflow\n";
        exit(-1);
    }

    else if (cnt==1)
    {
        tail=new Node;
        tail->content=i;
        tail->next=NULL;
        head->next=tail;
        cnt++;

//        cout<<BOLDGREEN<<"PUSH SUCCESSFULLY\n";

    }
    else
    {
        Node *p=new Node;
        p->content=i;
        p->next=NULL;
        tail->next=p;
        tail=p;
        cnt++;

//        cout<<BOLDGREEN<<"PUSH SUCCESSFULLY\n";
    }
}

void Queue::deQueue(int &i)
{
```

```
    if (cnt==0)
    {
        assert(head==nullptr&&tail==nullptr);
        cout<<BOLDRED<<"WORNG:---queue-empty\n";
    }
    else
    {
        cnt--;
        Node *p=head;
        i=p->content;
        head=head->next;
        if (head==NULL)
        {
            tail=NULL;
        }
        free(p);

//        cout<<BOLDGREEN<<"POP SUCCESSFULLY\n";
    }
    return;
}

void Queue::printAll()
{
    if (cnt==0)
    {
        cout<<BOLDYELLOW<<"Nothing to display\n";
    }
    else
    {
        for (Node *p=head;p!=nullptr;p=p->next)
        {
            cout<<BLUE<<p->content<<"\n";
        }
    }
    return;
}

#endif

void test()
{
    Queue queue;
    while (1)
    {
        int a;
        cin >> a;
        if (a == 0)
        {
            int b;
            cin >> b;
            queue.enQueue(b);
            queue.printAll();
        }
        else
        {
            int b;
            cin >> b;
```

```
            queue.deQueue(b);
            queue.printAll();
        }
    }
}

int main()
{
    Queue queue;
    int i = 0;
    printf("i=%d\n", i);
    queue.enQueue(1);
    queue.enQueue(2);
    queue.deQueue(i);
    queue.enQueue(3);
    printf("i=%d\n", i);
    queue.printAll();
    return 0;
    // test();
}
```

```
//MAKEFILE
Queue.o:
    g++ Queue.cpp -o $@

clean:
    rm -rf *.o

run:
    ./Queue.o
```