论 OI 编程与实际开发的区别

——写给经历过 OI 并且想继续开发下去的战友们

千里冰封

我是一个边做 OI 边做开发的不良高一青年，先经历了自学、自己进行开发工作的一系列工作，然后又在同学的影响下参加了信息学奥赛。在这里就个人经验胡乱写一些简单的文字。

OI(Olympic of Informatic)是信息学竞赛的简称，属于我大天朝的五大竞赛之一。这个东西将编程带进了咱学生党的生活，大大推广了编程这个东西，却也误了很多无知青年。许许多多 OI 风格的代码在实际开发或者工作岗位（这里显然说的是码农）中是不被推崇的。本文主要讲解一些本人在边搞 OI 边搞开发中获取的经验，由浅入深，从一些低层次的代码风格、编码习惯等开始，到算法、数据结构的模型设计等，分享出来，让更多 OIer 对于“编程”这个东西有更清醒的认识，也可以让各位做实际开发的朋友们了解一下 OI。当然我是指学生党。。

注：本文主讲 C 语言，以及一些 C++类与对象的使用。不讲 C++的类库设计。作者的主语言是 Java，其他语言有 C、lisp 等，不会 Pascal，不会 Visual Basic 这些~~乱七八糟~~在 OI 或者初等教育中用到的语言。所以我无法讲到这些语言。。至于我中文编程 E 语言，这个~~由于乌龟一样的运行效率和“学了之后连中文都不会用了”的谜之口碑，~~我从一开始就没考虑讲这个。。

事先声明，作者是 JetBrains 铁杆粉丝，所以实例代码都是用 CLion 写的，而 mingGW 我直接用的 DEV C++的 mingGW。虽然我也知道 Emacs、Vim 的厉害（而且我也用 Emacs 的），请不要因为代码编辑器而吐槽我！蟹蟹合作！

首先我们来看一些可能会在 OI 代码中看到的 main 函数写法：

```
int main() {
    return 0;
}
```

```
void main() {
    return ;
}
```

```
main() {
    return 0;
}
```

```
int main(void) {
    return 0;
}
```

实际上，在开发中更多使用的是这种写法：

```
int main(int argc, char* argv[]) {
    return 0;
}
```

这是因为 C 语言经常和命令行扯上关系，而命令行执行一个程序时，可以给一些参数，比如（命令是随手打的，不具实际意义）

```
~$ ./example.exe -s -f
```

这样，你的 main 函数中的 argc 就会是 3（因为命令行一共三个语句），并且：

argv[0] == “./example.exe”；argv[1] == “-s”；argv[2] == “-f”.

这样用户就能在执行程序时直接输入参数，而不是执行了程序之后再去 scanf 或者 std::cin。事实上这是一种非常流行的写法，不信你试试：

Windows： shutdown -s -f -t 00

Linux： sudo poweroff

。。。如果你真试了请不要打我。。

然后在 C99 标准中 main 函数的返回值必须是 int，这也是后面那句“return 0”存在的原因。多数情况下，操作系统会在一个程序停止执行之后检查它的返回值，返回 0 表示程序正常退出。比如你写一个 C 程序：

```
int main(int argc, char* argv[]) {
    int* a = (int *) 0x01;
    *a = 0;
    putchar(*a);
    return 0;
}
```

学过指针的都知道这是明显的作死吧。这可是一个教科书式的反面教材，大家千万不要这样使用指针。在 CLion 中编译运行，我在第四行输出指针 a 指向的 int 的值，如果程序正常执行（第三行的赋值成功）的话，程序应该会有回显，并且返回 0。但是运行结果如下：

```
C:\Users\user\.CLion12\system\cmake\generated\8da5f630\8da5f630\Release\clion_work_space.exe


Process finished with exit code 255
```

看！程序没有回显，并且返回了 255。

这里提到的两个细节 ,就是 main 函数的参数和返回值 ,它们都是有着重要意义的，但是它们在 OI 中由于没有作用（所有 IO 都是在 stdin 和 stdout 中进行，评测工具也不会检查程序返回值），所以常常被忽视。而这在实际开发中是异常重要的，请不要忽视它！

第二点，为大家讲解一个常见的误区。

由于 OI 中没有“工程”的概念，许多 OIer 认为，所谓“头文件”（.h 文件）就是函数库，里面放有一些函数，如果你想看一个标准库函数的实现的话只需要看看头文件里怎么写的就行了。并且在他们心中，“头文件”就只有 stdio.h、iostream、stdlib.h、math.h 等等，还有 STL 里面那些，还有很多很多。

实际上头文件只是一个放声明的地方，并且在实际开发中，每个.c 或者.cpp 文件都应该拥有它对应的头文件。一个头文件应该以预处理代码、函数声明和数据类型声明为主（typedef 等）。当然，在 C++引入了 OOP 的情况下，还有 class 的声明，等等。至于具体的实现则是应该放在同名.c 或者.cpp 文件中。这样的代码在编译时就只需要编译出目标文件，目标文件扩展名是.o，是二进制的机器码，然后将含有 main 函数的.c .h 组合和所有的.o 文件一起编译。这正是所谓“工程“的思想，OOP 带来的封装性将代码变得更加简洁，更适合多人合作开发项目。

关于头文件的趣闻真是屡见不鲜，我甚至看到过有些新手在头文件里写了个 main 函数，把我吓了一大跳。。

第三点，是一个老生常谈的话题：全局变量的使用。

话说本人是开发在先 ,OI 在后。当我第一次看 DFS 时 ,那时我看的是啊哈磊的《啊哈！算法》（除了代码风格有点难看之外，这是一本相当适合入门的好书），这个人幽默风趣，讲解代码语言通俗，就是代码风格有很多槽点，后文还会提到。他的 DFS 是举的一个地图寻宝的例子（貌似是，有点没印象了），然后我一看他的代码，第三行声明的那个全局变量 int a[101][101]对我幼小的心灵造成了难以磨灭的伤害。我发代码，大家感受一下（具体实现已经被我遗忘了，这里给出大致框架）：

```
#include <stdio.h>
#include <stdlib.h>
int a[101][101];
```

```
void dfs(int x, int y,int step);
int main(){
    return 0;
}
```

那时我坚定的认为使用全局变量是编程时的禁忌(现在我则更多地把它当做一个坏习惯，不像当初那么排斥），当时我盯着那个全局变量看了一节课（老师对不起！我应该认真听讲的），恨不得把那一页的代码撕下来。我只花了 30 秒就想到了不用全局变量的实现方法，但是书上就那么写着！书上就那么写着！书上就那么写着！我浑身难受！

要说在 OI 中这种做法有什么不好，确实，它不能带来什么明显的弊端。而且它还有自动初始化的好处，这在某种意义上也简化了代码。但是在实际开发中，使用全局变量的确是一个坏习惯。首先不说变量屏蔽的问题，这种用法很不安全，优秀的程序讲究一个封装，那么就一定不能有全局变量的出现。如果实在需要一个生存期和作用域都保持全局的变量的话，宁愿退而求其次使用静态本地变量 static int a[101][101]，也不要去最上面声明一个全局变量！谨记！

第四，这是一个细微的问题。本来这种用法并不常见，但是看到啊哈磊都这样了我觉得还是有必要写一写。

比如，声明一个二维数组并且手动初始化。注意是手动。

我的代码：

```
int main(int argc,char * argv[]){
    const int size = 100;
    int a[size][size];
    for(int i = 0; i < size; i++){
        for(int j = 0; j < size; j++)
            a[i][j] = 0;
    }
    return 0;
}
```

这是一个十分常规的方式，它是如此简单，以至于我想说明都不知道该怎么说。

而啊哈磊会这样写：

```
#define size 101
int a[size][size];
int main(int argc,char* argv[]){
    int i,j;
    for(i = 1; i <= size; i++){
        for(j = 1; j <= size; j++)
            a[i][j] = 0;
    }
    return 0;
}
```

多说无益，大家自己感受。忽略 for(int i = 0;;)的区别，我用的是 C99。

数组从 0 开始用是传统，请不要在无谓的地方标新立异。。尤其是在团队合作的时候，不良的代码风格会让你的同事们很崩溃。

第五，关于变量和函数命名规范。

首先大家应该知道的四种命名法（首先把变量的意义（名词）、方法或函数的功能（动宾短语）完整排列出来）：

1、 驼峰命名法：第一个单词首字母小写，后面的全部首字母大写。一般用于变量命名和方法命名。例（这是一段 Java 代码）：

```
if (!initialized) {
    initialized = true;
    configManager.initFromCameraParameters(theCamera);
    if (requestedFramingRectWidth > 0 && requestedFramingRectHeight > 0) {
        setManualFramingRect(requestedFramingRectWidth,
                requestedFramingRectHeight);
        requestedFramingRectWidth = 0;
        requestedFramingRectHeight = 0;
    }
}
```

2、 Pascal 命名法：全部首字母大写。一般用于类名，少数时候用于方法、函数命名。例如（代码有删减，是 Java）：

```
import android.content.Intent;
import android.graphics.drawable.Drawable;
import android.os.Build;
import android.os.Bundle;
import android.support.design.widget.NavigationView;
import android.util.Log;
import android.view.*;
import android.widget.LinearLayout;
import util.BaseActivity;
import util.PreferenceDataManager;
import java.util.ArrayList;
public class NewMainActivity extends BaseActivity
        implements NavigationView.OnNavigationItemSelectedListener {
    private PagerAdapter adapter;
    private LayoutInflater lf;
    private ArrayList<View> views;
    private ArrayList<String> titles;
    private PreferenceDataManager manager;
    private LinearLayout background;
}
```

3、 匈牙利命名法：在 Pascal 的基础上，在前面再加上一个小写字母。不同的小写字母代表不同的意义。一般用于变量命名。尤其是类的成员变量或者局部变量。例如（偷个懒，把前面的稍作修改，是 C 语言的）：

```
#define size 100
int map[size][size];
int main(int argc,char* argv[]){
    int mHorizontal, mVertical;
    for(mHorizontal = 0; mHorizontal < size; mHorizontal++)
        for(mVertical = 0; mVertical < size; mVertical++)
            map[mHorizontal][mVertical] = 0;
    return 0;
}
```

4、 下划线命名法：顾名思义，单词全部小写，中间用下划线隔开。这个用途很多，比如包名、Android 中的资源文件名、C/C++语言的宏名和自定义数据类型名、etc.例如（摘自神圣的 stdio.h）：

```
#undef __MINGW_PRINTF_FORMAT
#undef __MINGW_SCANF_FORMAT
#define __MINGW_PRINTF_FORMAT    ms_printf
#define __MINGW_SCANF_FORMAT    ms_scanf
#undef __builtin_vsnprintf
#undef __builtin_vsprintf
  int __cdecl fprintf(FILE * __restrict__ _File,const char * __restrict__ _Format,...);
  int __cdecl printf(const char * __restrict__ _Format,...);
  int __cdecl sprintf(char * __restrict__ _Dest,const char * __restrict__ _Format,...)
__MINGW_ATTRIB_DEPRECATED_SEC_WARN;
  int __cdecl vfprintf(FILE * __restrict__ _File,const char * __restrict__ _Format,va_list _ArgList);
  int __cdecl vprintf(const char * __restrict__ _Format,va_list _ArgList);
  int __cdecl vsprintf(char * __restrict__ _Dest,const char * __restrict__ _Format,va_list _Args)
__MINGW_ATTRIB_DEPRECATED_SEC_WARN;
  int __cdecl fscanf(FILE * __restrict__ _File,const char * __restrict__ _Format,...)
__MINGW_ATTRIB_DEPRECATED_SEC_WARN;
  int __cdecl scanf(const char * __restrict__ _Format,...) __MINGW_ATTRIB_DEPRECATED_SEC_WARN;
  int __cdecl sscanf(const char * __restrict__ _Src,const char * __restrict__ _Format,...)
__MINGW_ATTRIB_DEPRECATED_SEC_WARN;
```

看到了吧？

使用以上的四种规范命名（这四种是国际通用的命名法~~，少跟我来什么拼音，我不认识你~~）让阅读你的代码的人顾名思义，也方便了维护，而在 OI 中，变量名一般都是 a,b,c,d,i,j,cnt,idx，函数名一般都是 dfs,bfs,~~fuck,fuckzk~~ 等，由于 OI 代码千行级的还是很少，不需要考虑维护困难的问题，所以写出这样的命名情有可原，咳咳。

第六，关于算法设计。

一般在 OI 题中你可以看到一个明晃晃的”时间限制：1s“，但是空间限制却是“G”这个级别的。这就证明，OI 是几乎不看空间复杂度的。而时间复杂度，就变得尤为重要。但是在实际开发中，你更是两方面都要考虑。但是实际开发中的要求就不那么严格，作为一个 Javaer，经常用到 ArrayList，我自认为这个容器很高效，事实上我从未因为”哪个容器效率低“这种问题而困扰。但是当我接触了 OI 之后发现，在做一些题时，我用了 List，得 8 分，但是换成 vector，就是 9 分，如果纯数组实现，就 AC 了，这显然对于效率问题卡得非常紧。而 OI 很少在意空间复杂度的做法也对我有一定影响，比如我有个使用 SQLite 作为数据库的 APP ,我使用了表之间的映射的方法来表示类别。这种一看就知道空间复杂度高的吓人的方法我却不以为然地使用了，当然这是下策。。

我还有一个城堡 RPG 冒险的文字游戏，这个游戏的地图系统中使用了一个 Room 类一个 RoomMap 类，一个是房间单元，一个是地图。目前游戏中有 20 个房间，每个房间里面都有一个 ArrayList 一个 HashMap，这样的设计我刚刚做的时候（我是先开发后 OI 的）觉得无可厚非，但是学了 OI 之后，我才发现这是一种极其不好的做法，数据规模一旦变得巨大无比的时候，容器越多越容易爆。所以我不惜 3 个晚上的时间把所有房间的 ArrayList 全部优化成一个 int 数组，然后建了个表来映射。这大大减少了时间复杂度和内存占用，虽然我这个内存本身连 1M 都没有。。。

OI 本身追求的就是一个算法优化，因为出题老师永远都会想方设法地制造一些数据量极其庞大的测试数据。而你的时间复杂度高了，就会丢分。

第七，是一个小事，常量推荐大家用 const int，而不是 define，因为常量就常量，不要搞成宏，这样给人的感觉不舒服。

最后，关于数组的大小。很多题说，先输入一个数 N，N<1000，然后再输入 N 个数。这种情况大多数 OIer 的思路是这样：

```
#include <stdio.h>
#define MAX 1000
int main(int argc,char* argv[]){
    int a[MAX];
    int N;
    scanf("%d", &N);
    for(int i = 0; i < N; i++){
        scanf("%d", &a[i]);
    }
    return 0;
}
```

这种做法其实并不好。因为如果 N<<MAX 的话，就会浪费一大堆空间，而如果（未说明 N 的范围的情况下）N>=MAX 的话，那岂不是会 IndexOutOfBoundException？开个玩笑，这是 Java 的异常，意思就是数组访问越界了。233

这时我推荐大家一个做法。就是 malloc。

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

int main(int argc,char* argv[]){
    int *a;
    int N;
    scanf("%d", &N);
    a = (int *) malloc(N * sizeof(int));
    for(int i = 0; i < N; i++){
        scanf("%d", &a[i]);
    }

    free(a);
    return 0;
}
```

看到了吗？首先使用 malloc 函数，参数是内存大小，这里我们给出 N 个 int 的大小，即 N * sizeof(int)，然后再强制转化为 int 指针型。malloc 缺省返回 void 指针型。

这种做法就是刚好从内存中获取了所需要的大小的空间。最后再用 free 函数释放相应内存。这就避免了因为数组大小不适当而出现的问题，虽然我也渐渐习惯了开一个巨大的数组。。

好了，以上就是我目前~~废话~~总结的一些关于 OI 和实际开发之间的区别。

2016 年 2 月 8 日