2012“种子杯”编程 PK 赛 复赛说明文档

编译器功能拓展尝试

队名 xxx	光学与电子信息学院
队员 徐哲钊	电子科学与技术 1004 班
队员 徐洁	电子科学与技术 1003 班

目录

一、编译环境说明

本部分介绍程序的开发的环境，以及从整体上叙述程序工程目录下面的源代码的作用。

1.1 开发测试环境

使用的编译器为 Visual Studio 2008 英文版和 Visual Studio 2010 中文版。程序的开发、测试、代码整理、文档编写均在 win7 下面完成。

1.2 程序代码目录

为了使程序更加简洁易懂，将每个函数的头文件和源文件分开，并且文件名一一对应。其中头文件声明源文件中需要使用的函数，对应的源文件中则定义了函数的功能。

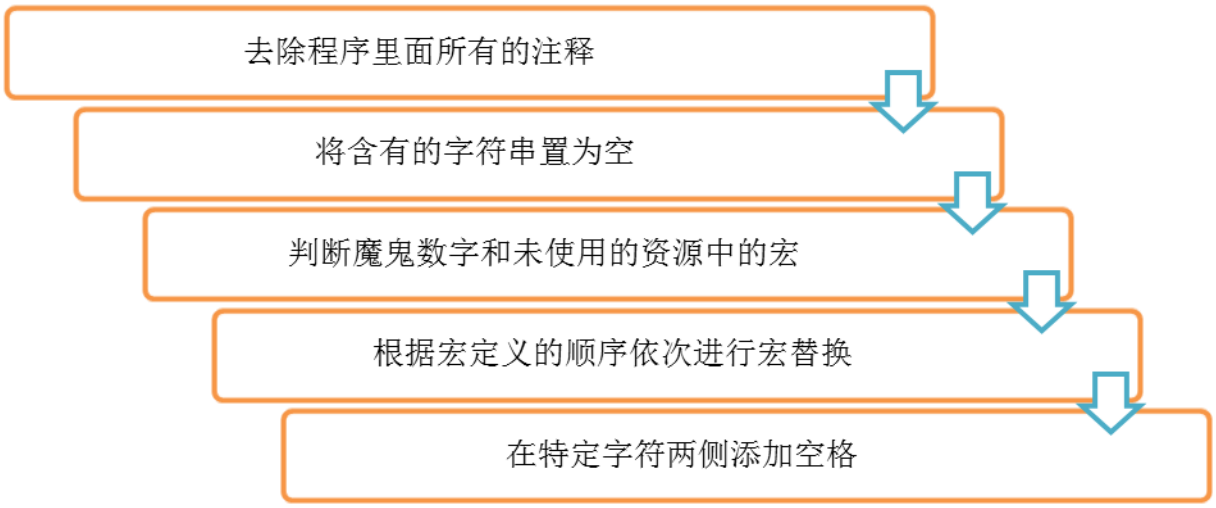
功能点	header、source 文件名	函数作用
防止错误赋值	5_1	检测错误赋值
指针归空	5_2	指针归空检查
无效分支	5_3_1	检查 if 语句
	5_3_2	检查 while 语句
	5_3_3	检查 switch 语句
	5_3_4	检查 for 语句
魔鬼数字	5_4	检查魔鬼数字
未使用的资源	5_5_1	函数未被调用过
	5_5_2	宏未被使用
	5_5_3	变量未被使用
	5_5_4	函数返回值未被使用
函数关系调用图	5_6	打印函数关系图
自定义	5_7_1_Custom	检查内存泄漏
	5_7_2_Custom	检查变量使用前是否被赋初值

二、程序的结构

2.1 命令行预处理

为了方便对命令的读取和处理，首先对命令进行格式化处理。由于格式化时需要替换宏，但是程序中需要检查到宏、需要考虑魔鬼数字，宏未被使用的情况，所以应该首先对魔鬼数字和宏未被使用的情况进行判断，之后再进行宏替换。

具体的格式化步骤为



其中特定字符包括
"(" ")" "," ";" "<" ">" "=" "!" "\" "*" "+" "-"
其中如果出现多个组合一起，则只在组合的字符两侧添加。例如出现">="时只在字符两侧添加，不必再中间添加空格。
这样处理的优势是方便读取命令和字符串：只需要根据";"，读取一行命令，根据空格读取字符串或者关键字。

2.2 各功能点的实现

程序中的全局变量为

全局变量名	数据类型	作用
command	string 数组	储存命令
cmdCount	int	命令行的数目
errLen	int	错误的个数
errNum	int 数组	每一个错误出现的行数
errType	int 数组	每个错误的类型

按照需求依次实现每一个功能点，详细见程序的模块。按照顺序检查到各个错误，然后根据错误出现的行数从小到大排序、打印。

三、程序的模块

下部分分别详细介绍了各个功能实现的过程。由于程序的实现分为很多函数来实现，不可能一一细说每个函数的功能，所以在此介绍各个功能点实现的思路。

3.1 防止错误赋值

错误赋值的情况仅当"=="号两端左端出现变量，右端出现常量时候报错。因此只需要依次检查"=="符号两端的变量类型，如果出现该情况则报错。其中右端为函数返回值时不需要报错，左端为数组类型的变量时需要报错。

3.2 指针归空

由于每一个指针会申请地址空间，空间地址最终会被释放，因此首先查找 free 关键字；然后提取 free 后面的字符，检查是否出现给该字符赋值等于 NULL 的情况。如果未出现则报错。

3.3 无效分支

考虑到无效分支的情况比较复杂，我们将情况分解，采取从简单到复杂的方法。分为 if，while，for，switch 来实现。

当出现上述关键字时候，提取括号中的变量名，然后依次向上查找是否存在给变量赋定值的情况，如果存在则报错。

分支类型	实现步骤
if，while	1、提取 if，while 后面的变量
	2、向上检测该变量是否被赋定值
	3、若存在则报错
for	1、提取 for 语句后面第二个分号前的变量
	2、向前检测变量的值
	3、判断 for 语句的循环条件
	4、若恒不成立则报错
switch	1、提取 switch 后面的变量

	2、得到变量值
	3、在 case 不成立的地方报错

3.4 魔鬼数字

判断魔鬼数字是在宏替换之前，只需要查找除去宏定义以后，程序中出现的所有数字，如果不为-1,0,1 则为魔鬼数字。

3.5 未使用的资源

将未使用的资源分为三类，如下

3.5.1 函数和宏

未使用的资源类型	出现的次数	报错条件
函数	函数名出现一次	函数只有声明或者定义，在该处报错
	函数名出现两次	有定义和声明，则没被使用，报错
		只有定义，无声明，则不报错
	函数名出现三次及以上	不报错
宏	宏出现一次	宏未使用，报错
	宏出现两次及以上	不报错

在对宏统计使用次数的时候忽略了宏未被使用的情况。

3.5.2 变量

分为每个函数单独实现，首先查找变量定义时出现的位置，然后再函数内部查找变量是否被使用。

在以下情况下变量被使用：

1. 自己被改写
2. 给其它变量赋值
3. 间接决定其它变量数值. eg b = p[a]中的 a

3.5.3 返回值

当函数返回值未被赋给其它变量时报错，而且函数前面没有（void）时，报错。

3.6 函数关系调用图

3.6.1 简单情况

得到 main 函数调用的所有函数（假设为 f1，f2），然后依次得到 f1，f2 里面调用其他函数的情况（假设 f1 调用了 f3,f4）,然后得到 f3，f4 的调用情况，直到函数里面不调用其他函数。

3.6.2 递归调用

在简单情况下，如果调用的函数在之前已经被调用，则停止，且判断为递归调用。

四、功能完成的情况

功能点都完成了，但是比较复杂的情况，
比如无效分支，一个变量可能会牵扯出很多其它的变量，这种情况我们未考虑。
错误赋值，右端为复杂表达式
未使用的资源，当表达式比较复杂时，可能会忽略一些未使用的资源

五、程序的亮点

1) 替换了字符串，避免了关键字和字符串部分匹配的错误情况
2) 变量声明可以放在程序的任何地方
3) 完美实现了带参数的宏对程序的影响
4) 注释可以出现在任何地方，对程序无影响

六、自定义的功能

测试时加入 –x 参数 形如 SeedCup.exe input.c –x output.txt

6.1 内存检查

检测内存是否泄漏，分为两种情况
1.1、malloc 开了内存，没有 free 这块内存。

1.2、用指针指向 malloc 开的内存，在没有释放这块内存之前改变了该的指针的值，使该块内存无法释放，不动态跟踪 free 指针的值，如下

```
int *p1 = (int)malloc(sizeof(int));
int *p2 = (int)malloc(sizeof(int));
p1 = p2;
free(p1);
```

虽然 p2 的内存块由于 p1 的原因释放了，但是还是要在 p2 的声明处报错。

报错位置：在 malloc 内存的位置

错误代码：7

6. 2 未赋初值就使用

检测变量是否未赋初值就使用了

如：int i;

```
int j = i;
```

报错位置：在声明变量的地方报错

错误代码：8

待改进：检查数组使用前是否赋初值还没有实现

在本文档的最后给出了测试的样例。同时目录下含有.c 格式的测试文本。

七、参加种子杯的感受

没有参加过“种子杯”，你还好意思说自己是程序员？这是种子杯卡贴上的一句话。确实，种子杯的比赛比赛难度、工作量都很大，比赛围绕算法和数据结构，考验我们的编程能力。

但是，我想说的是，对于高中没有任何编程基础、大学不想当程序员、没学习过数据结构的我和队友，凭借兴趣和辛勤付出，我们同样可以在这样一个比赛中披荆斩棘、崭露头角。

比赛的六天时间是我们最累的时候，各种实验、各门天书一样的课，一般人也许很难抽出更多时间。但是面对复杂的编程题，我们没有丝毫的懈怠和放弃，每天晚上熄灯以后是我们最兴奋的时候，为了调试一个小错误我们经常一起工作到两三点而毫无睡意。在我们的共同努力下，最终我们可以拿出一份像样的作品。在别人看来也许是漏洞百出，但是对我们来说这份作品却饱含我们全力完成一件自己感兴趣的事的汗水。

比赛的奖金确实非常丰厚，无论谁看了都会动心。但是参加比赛的过程才是最重要的，无论是否能够拿到奖金我觉得我们心中对自己的成功或者失败都已经有了一个答案。

附录

测试自定义功能 1

测试文件 1

```
int main()
{
	int *p = (int)malloc(sizeof(int));

	return 0;
}
```

测试文件 2

```
int *p = (int)malloc(sizeof(int));;
int main()
{
	return 0;
}
```

测试自定义功能 2

测试文件 1

```
int main()
{
	int i;
	int j;
	j = i;
	return 0;
}
```

测试文件 2

```
int g;
int main()
{
	int i;
	i = g;
	return 0;
}
```