Report1

Debug

由于最初的测试程序相对简单，且我们并没有考虑诸如两个`#pragma overflowCheck`对应一个函数或者违反C语言语法规则和书写惯例（函数给出定义后无需声明等），因此并没有发现问题。

```
#pragma overflowcheck
int func1(void){
    return 0;
}

int func2(void) {
        return 1;
}

#pragma overflowcheck

int func3(void){
        return 0;
}

int main(){

        func3();
        return 0;
}
```

但随后在对先定义后声明的情况进行测试时，出现了严重的问题：

```
llvm1@teacher-PowerEdge-M640:~/llvm-install/bin$ ./clang -cc1 -load ~/build/lib/TraverseFunctionDecls.so -plugin traverse-fn-decls ~/example.c
func1: 1
func2: 1
func3: 1
func4: 1
main: 0
llvm1@teacher-PowerEdge-M640:~/llvm-install/bin$ cat ~/example.c
#pragma overflowcheck
int func1(void){
    return 0;
};

int func2(void) {
        return 1;
}

#pragma overflowcheck

int func2();

int func3(void){
        return 0;
}

int main(){

        func3();
        return 0;
}
#pragma overflowcheck
int func4(void){
        return 0;
}
```

可以注意到，尽管`func2`在第二个`#pragma overflowCheck`关键字之前便已经定义，距离第二个`#pragma`最近的函数定义是`func3`，`func2`理应不被越界检查，但输出却是1，更直观的说，按照guide book的要求，一个关键字最多匹配一个函数（但一个函数能匹配多少`#pragma overflowCheck`并没有要求），此时被标记为需要进行越界检查的函数最多只有三个，但最后却输出了四个1，这显然有严重的问题。

由于此工程庞大，流程复杂，贸然使用gdb必然会导致极其复杂的工作量，此时我选择多使用几个测试文件判断错误可能发生的地方。

首先针对第一个错误，起初我认为和不符合C语言的编写标准相关（`func1`后的分号），在去掉分号后的确执行出正确的结果，但其他的测试表明，问题并不是那么简单。

```
llvm1@teacher-PowerEdge-M640:~/llvm-install/bin$ ./clang -cc1 -load ~/build/lib/TraverseFunctionDecls.so -plugin traverse-fn-decls ~/example.c
func1: 1
func2: 1
func3: 1
func4: 1
main: 0
llvm1@teacher-PowerEdge-M640:~/llvm-install/bin$ cat ~/example.c
#pragma overflowcheck
int func1(void){
    return 0;
}

int i = 0;

int func2(void) {
        return 1;
}

#pragma overflowcheck

int func2();

int func3(void){
        return 0;
}

int main(){

        func3();
        return 0;
}
#pragma overflowcheck
int func4(void){
        return 0;
}
llvm1@teacher-PowerEdge-M640:~/llvm-install/bin$
```

上图的测试程序并没有任何语法上的问题，但依然出现了同图一一样的问题，而且由于其能通过一些简单测试，故可以暂时假定在paser过程（抓去Token）并没有出现问题。

```
main: 0
llvm1@teacher-PowerEdge-M640:~/llvm-install/bin$ ./clang -cc1 -load ~/build/lib/TraverseFunctionDecls.so -plugin traverse-fn-decls ~/example.c
func1: 1
func2: 1
func3: 1
func4: 1
main: 0
llvm1@teacher-PowerEdge-M640:~/llvm-install/bin$ cat ~/example.c
#pragma overflowcheck
int func1(void){
    return 0;
}

int i = 0;

int func2(void) {
        return 1;
}

int func2(void);

#pragma overflowcheck

int func3(void){
        return 0;
}

int main(){

        func3();
        return 0;
}
#pragma overflowcheck
int func4(void){
        return 0;
}
```

随后将第二个`#pragma overflowCheck`移至`func2`的声明后，发现依然出错，而在去掉func2的声明后，则执行结果正常。

```
llvm1@teacher-PowerEdge-M640:~/llvm-install/bin$ ./clang -cc1 -load ~/build/lib/TraverseFunctionDecls.so -plugin traverse-fn-decls ~/example.c
func1: 1
func2: 0
func3: 1
main: 0
llvm1@teacher-PowerEdge-M640:~/llvm-install/bin$ cat ~/example.c
#pragma overflowcheck
int func1(void){
    return 0;
}

int i = 0;

int func2(void) {
        return 1;
}

//int func2(void);

#pragma overflowcheck

int func3(void){
        return 0;
}

int main(){

        func3();
        return 0;
}
//#pragma overflowcheck
//int func4(void){
//        return 0;
//}
```

因此可以提出猜想，`isOverflowCheck`有一个初始化的过程，在扫入Token前其应当初始化为false，换言之，对其的运行前准备不应当只有位宽的定义，还应当有false的初始化，因此使用gdb，在对应函数处下断点，之后执行，发现其初始化果然有问题。

随后参照其余is标识位标准，对其进行false的初始化。

```
                                                    StartLoc),
DeclContext(DK),
ParamInfo(0), Body(),
SClass(S),
IsInline(isInlineSpecified), IsInlineSpecified(isInlineSpecified),
IsVirtualAsWritten(false), IsPure(false), HasInheritedPrototype(false),
HasWrittenPrototype(true), IsDeleted(false), IsTrivial(false),
IsDefaulted(false), IsExplicitlyDefaulted(false), IsOverflowCheck(false),
HasImplicitReturnZero(false), IsLateTemplateParsed(false),
IsConstexpr(isConstexprSpecified), HasSkippedBody(false),
EndRangeLoc(NameInfo.getEndLoc()),
TemplateOrSpecialization(),
DNLoc(NameInfo.getInfo()) {}
```

随后继续gdb测试，发现其满足预期。

```
Thread 3.1 "clang-3.3" hit Breakpoint 5, clang::FunctionDecl::setOverflowCheck (
    this=0x5171a50, OverflowCheckOn=false)
    at /home/llvm1/llvm/tools/clang/include/clang/AST/Decl.h:1619
1619      void setOverflowCheck(bool OverflowCheckOn = true) { IsOverflowCheck = OverflowChec
kOn; };
(gdb) display OverflowCheckOn
1: OverflowCheckOn = false
```

由此解决此bug，此后不再有问题。