Разработка адаптивного стилевого анализатора программ на языке C++

Автор: Уваров Никита, 11 класс

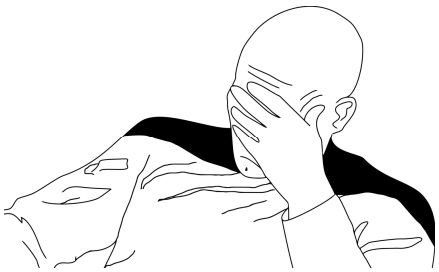
Научный руководитель: Дединский Илья Рудольфович

О проблеме

```
int dl=-3;
un[!p]=0;
int i0=i;
while(un[p]){
i++;
d[i] =  dl;
if(!last[i]){
u[!p][a[i]]=1+i, un[!p]++;
}
if(u[p][a[i]]){
v[a[i]] = u[p][a[i]]-1;
u[p][a[i]]=0, un[p]--;
}
}
dl = v[a[i]];
for(i0++;i0<=i;i0++){
d[i0] = dl;
v[a[i0]]=0;
}
p=!p;
```

```
for  (int i=0;i<=nc;i++)
    { (a[i]!=' ')
    printf ("%s ",a[i]);
    else
    {
        if (a[i+1]!=' ')
        printf ("%s ", a[i]);
        else
        for (int j=0;i<=j;j++)
        {if (a[i+j]==' ')
        {
        printf("%s ",a[i+1]);
        i=i+j;
        }
        }

        }
        }
```



Приведенные фрагменты – реальный код из доступных источников.
Форматирование сохранено.

Стиль форматирования

```
while (true) {
    if (counter > MAX_COUNT) {
        nLoops++;
        counter %= MAX_COUNT;
        if (nLoops > MAX_LOOPS)
            break;
    }
    counter++;
    doWork(counter);
}
```

*Стиль **K&R***

VS

```
while(true) {
if(counter>MAX_COUNT){
nLoops++;
counter%=MAX_COUNT;
if(nLoops>  MAX_LOOPS)
break;
}
counter ++;
doWork(counter);
}
```

«No style»

Хорошо:

- структура программы видна с первого взгляда

Плохо:

- нужно вглядываться, искать знаки препинания

Стиль именования переменных

```
while (true) {
    if (counter > MAX_COUNT) {
        nLoops++;
        counter %= MAX_COUNT;
        if (nLoops > MAX_LOOPS)
            break;
    }
    counter++;
    doWork(counter);
}
```

Имена понятны человеку

VS

```
while (true) {
    if (C > MC) {
        n++;
        C %= MC;
        if (n > ML)
            break;
    }
    C++;
    d(C);
}
```

Имена понятны лишь компилятору

Хорошо:

- легко понять назначение фрагмента кода

Плохо:

- назначение фрагмента кода не ясно без контекста
- легко перепутать (ML и MC)

Автоматическая проверка стиля

Существующие способы:

- cxxchecker, cpplint, Vera++, KWStyle
- сравнение с AStyle

Недостаток – фиксированный стиль (эталон проверки)

```
int main() {
    printf(“Hello”);
}
```

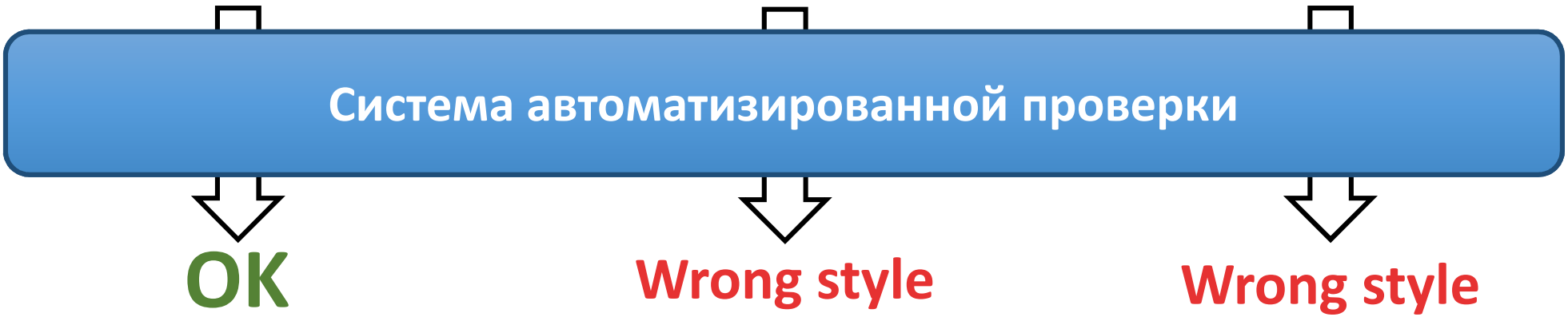
Стиль **K&R**

```
int main()
    {
    printf(“Hello”);
    }
```

Стиль **Whitesmiths**

```
int main()
{
    printf(“Hello”);
}
```

Стиль **Allman**



Цель работы

Разработать инструмент проверки стиля программы,
не требующий предварительного задания стиля,
а определяющий его автоматически.

Требования:

- *Адаптивность*, то есть возможность автоматического определения преобладающего стиля
- *Устойчивость* к стилистическим ошибкам и возможность их диагностики
- *Абстракция от конкретного языка* программирования (разделение на front- и back-end)

Сложности при анализе форматирования

- Однострочные конструкции (one-line statements)

```
if (nArguments != 2)  return 0;
if (cached[argument]) return retrieveCached (argument);
// обычно «return» располагается на отдельной строке
```

- Переносы (line breaks)

```
if (getHolidaysInMonth(month) + getWeekendsInMonth(month) >=
    getWorkDaysRequired(month) - getWorkTreshold(month) &&
    useExceptions) // перенос длинного условия оператора «if»
```

- Выравнивание похожих строк

```
const char* file_signature    = "BCDE";
const char* section_signature = "SE";    // выравнивание
const char* header_signature  = "HE";    // в виде таблицы
```

Сложности при анализе форматирования

- Использование препроцессора, комментарии

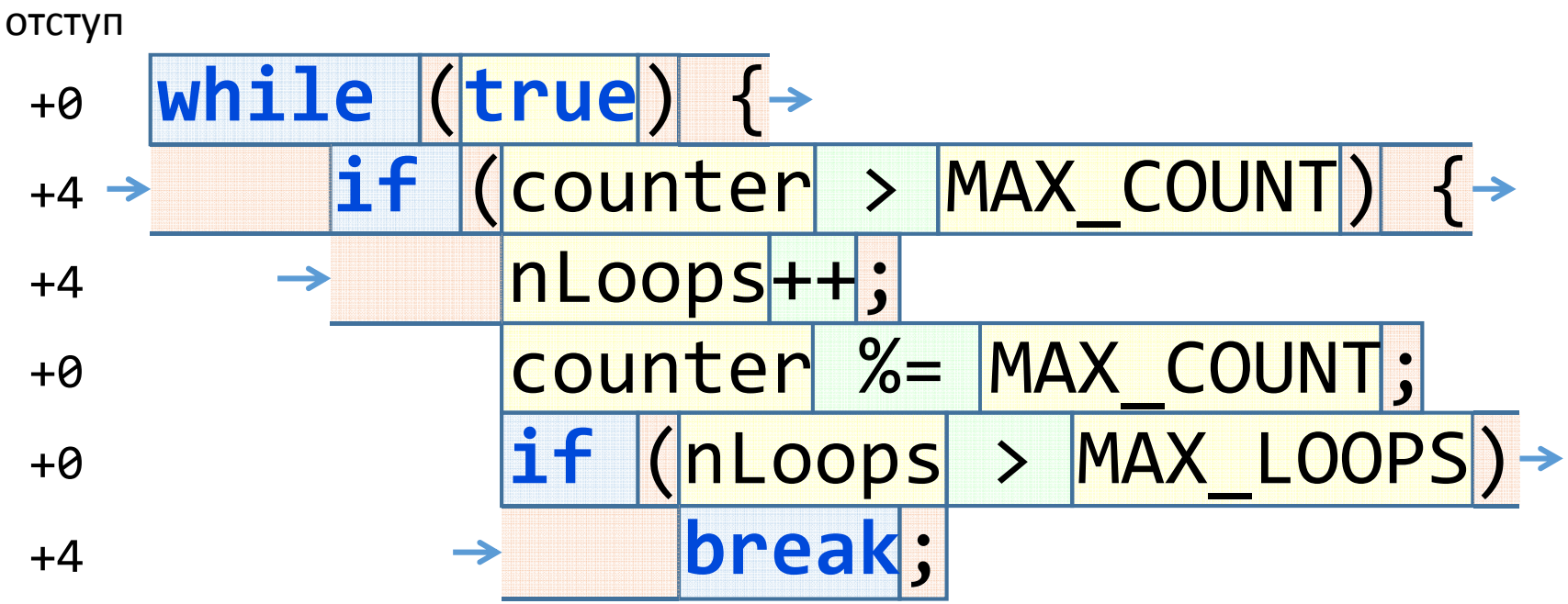
```
#define ifdebug(...) if(debugEnabled(__VA_ARGS__))
ifdebug(__LINE__)
{
      // использование препроцессора
      // для создания собственного условного оператора
}
```

- Расстановка пустых строк программистом

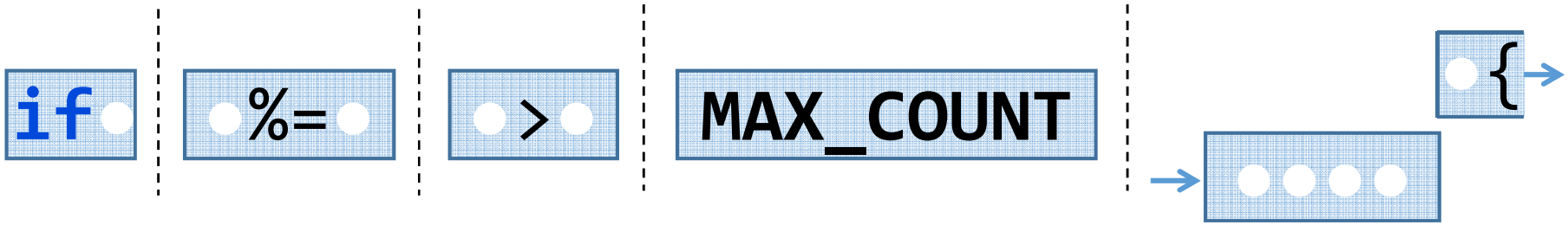
```
int nEntries;
scanf("%d", &nEntries);


int nRequests;              // предыдущая строка – пустая,
scanf("%d", &nRequests); // она отделяет логические связанные
                            // группы операторов
```

Модель анализа стиля форматирования



Каждый тип лексем вносит в форматирование фиксированный вклад:



Стилевые уравнения

```
if (counter > MAX_COUNT)
```

```
if.suffix          + l_paren.prefix    = 1
l_paren.suffix     + identifier.prefix = 0
identifier.suffix  + greater.prefix    = 1
greater.suffix     + identifier.prefix = 1
identifier.suffix  + r_paren.prefix    = 0
```

```
if (
   (counter
      counter >
            > MAX_COUNT
               MAX_COUNT)
```

```
if.suffix          = 1
l_paren.suffix     = 0
l_paren.prefix     = 0
identifier.suffix  = 0
identifier.prefix  = 0
greater.suffix     = 1
greater.prefix     = 1
r_paren.prefix     = 0
```

Невидимые модификаторы

Проблема:

```
if (variable != 0)
    callFunction(variable);
callOther();
```

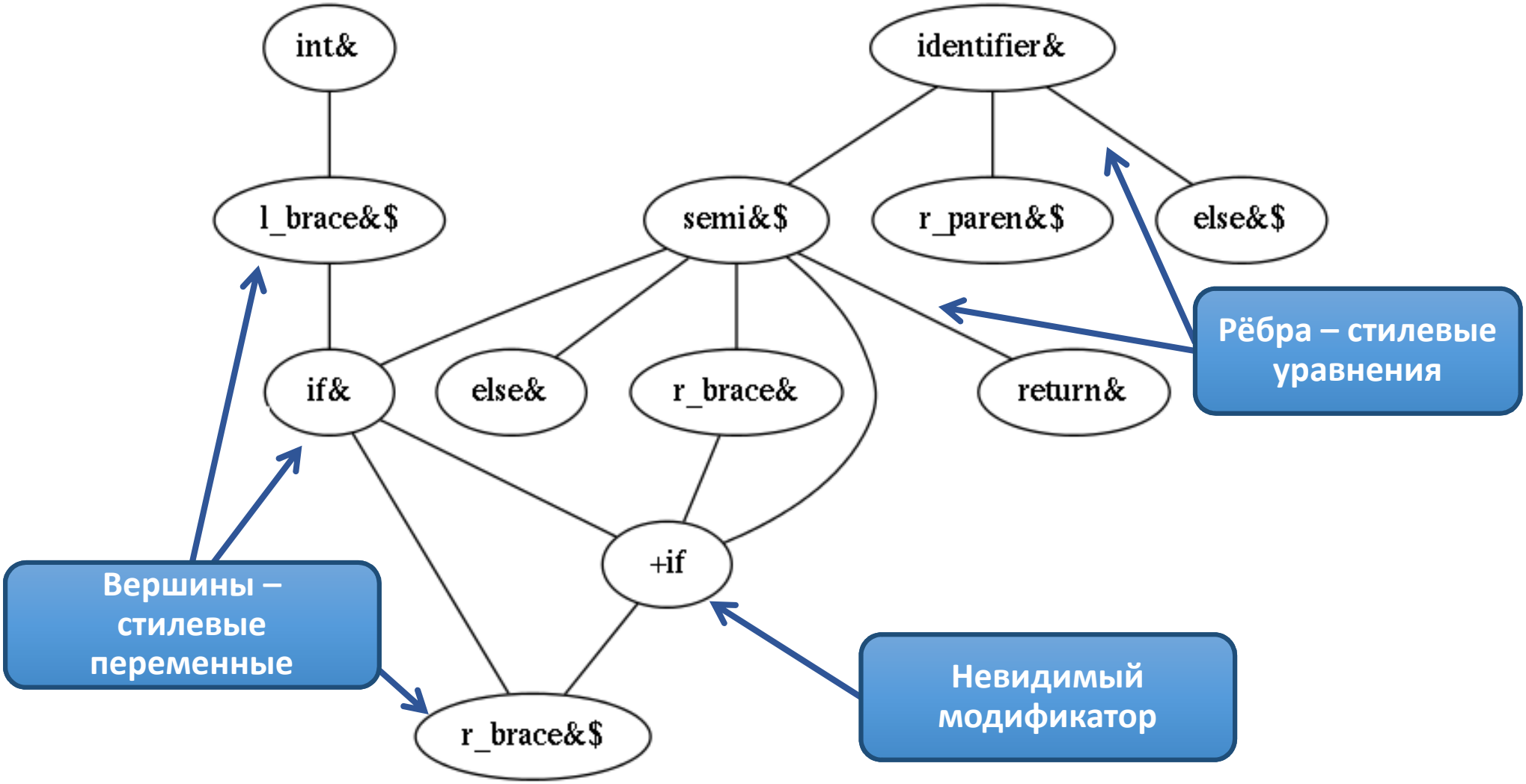
Изменение отступа **-4** (после лексемы) не обусловлено типами лексем
На стиль влияет синтаксическая структура программы (оператор **`if`**)

Решение:
Лексеме **`";"`** назначается модификатор конца области видимости

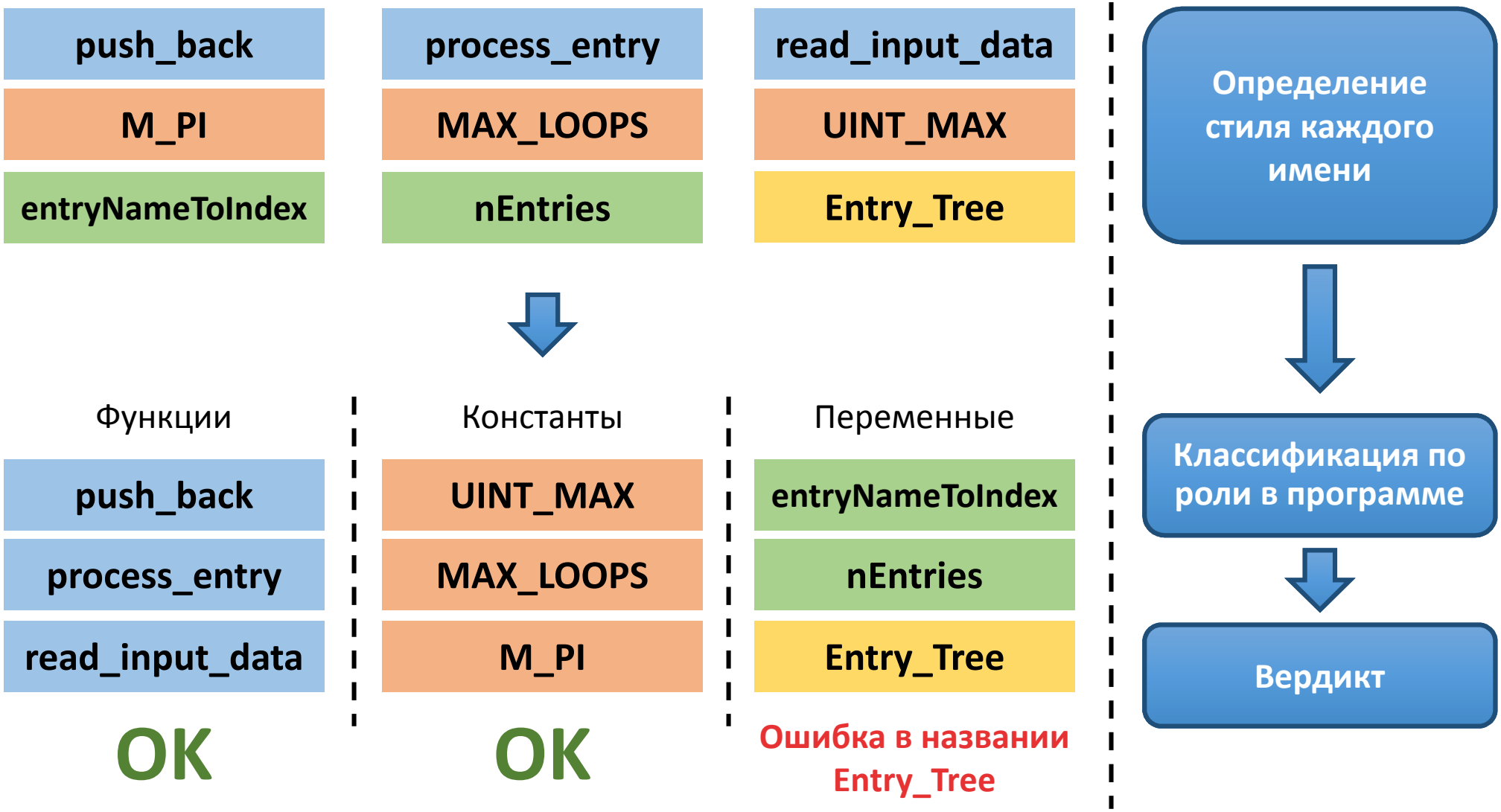
Новая проблема: нужен синтаксический анализатор

Решение стилевых уравнений

Метод “разделяй-и-властвуй” на графе зависимостей



Анализ имён переменных



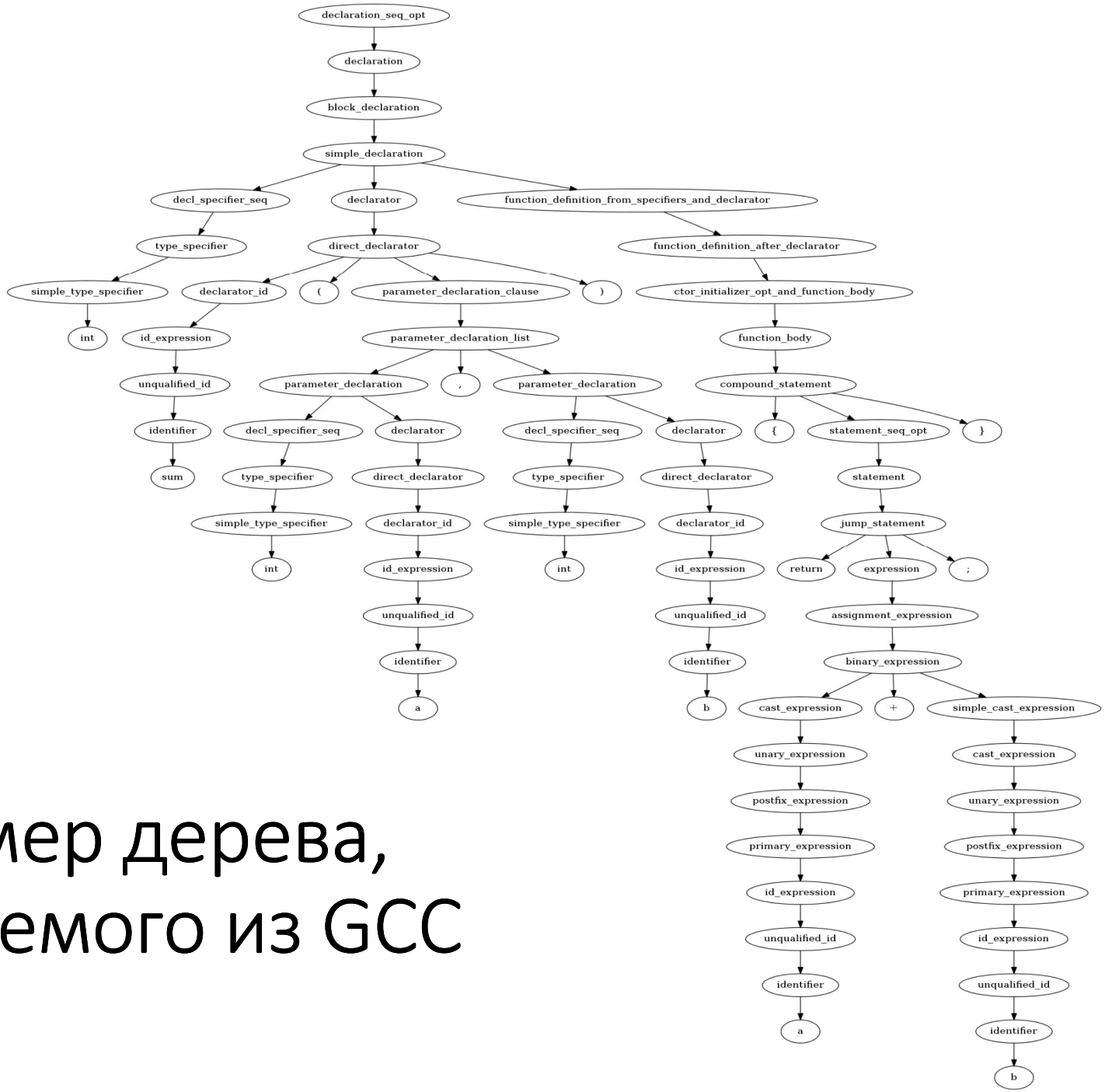
Необходимость синтаксического анализа

Синтаксическая информация нужна для:

- назначения невидимых модификаторов
- классификациии многофункциональных лексем (“ : ”)

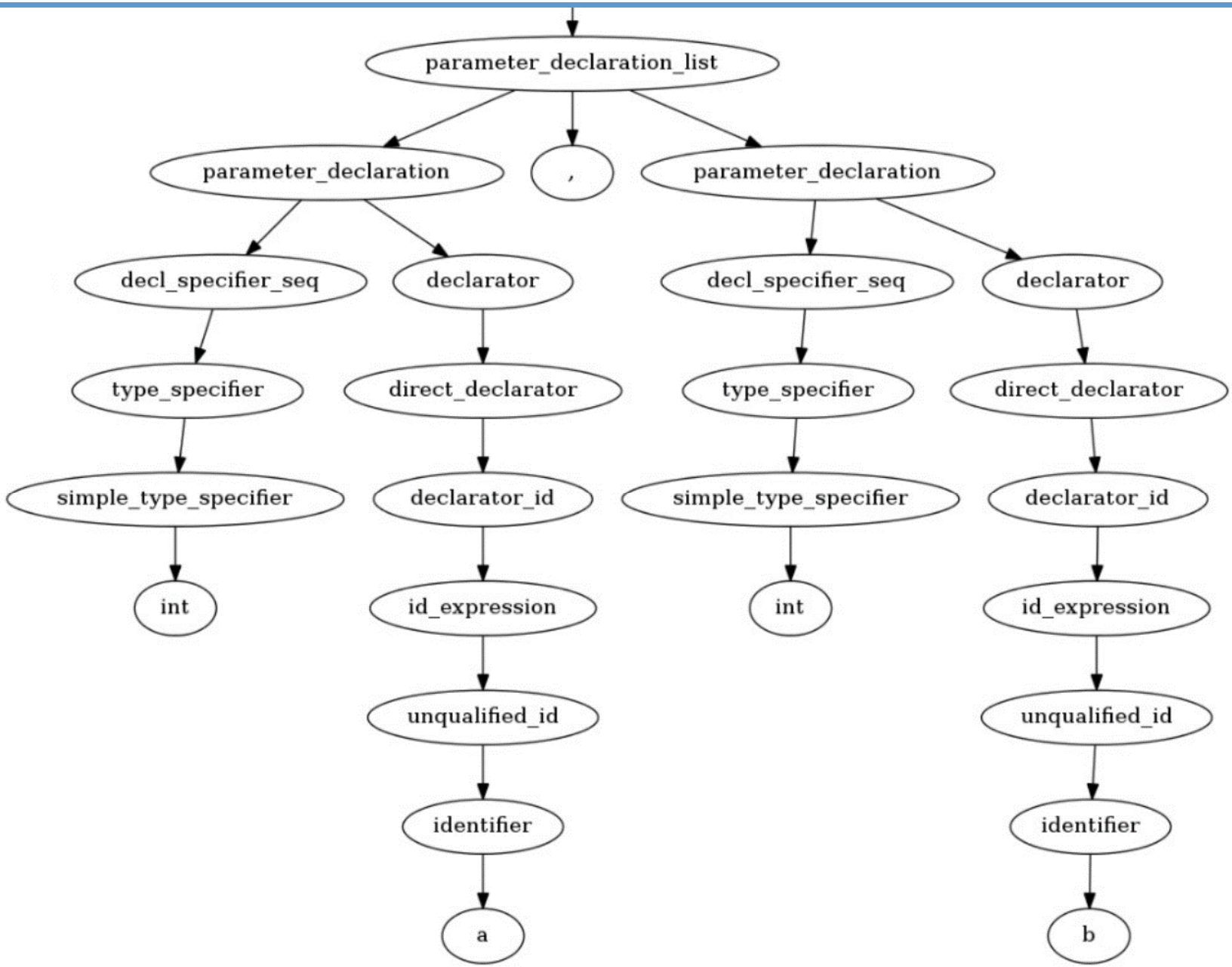
Рассмотренные способы получения (для C++):

Проект	Free Software?	Библиотека?
VivaCore	**Нет**	**Да**
GCC (g++ front-end)	**Да**	**Нет**
Clang (LibTooling)	**Да**	**Да**



Пример дерева, получаемого из GCC

Пример дерева, получаемого из GCC



Примеры работы анализатора

```
int main() {
    int a, b;
    scanf("%d %d", &a,&b);

    if (a != b) {
        if (a > b)
            a %= b;
        else
            b %=a;
    }

    if (a > 0 && b > 0)↪
    assert(!"impossible");

    printf("%d\n", a + b);
    return 0;
}
```

Стиль **K&R**

```
int main()
{
    int a = 5;

    if (a == 3)
        a++;
    else if (a == 4)
        a--;
    else
        if (a == 5)↪
    a *= 2;
    else
        a - 5;

    if (a == 6)
    {
    }


    if (a != 3)↪
    {
    }↪
↪
↪
    if (a++) {
    }↪
}
```

Смешение стилей

```
int main()
    {
    int a, b;
    scanf ("%d %d", &a, &b);

    if (a && b)
        {
        if (a>b)
            a %= b;
        else
            b %= a;
        }

    if (a>0 && b>0)
        assert (!"impossible");
    else
        {
        printf("GCD is ");
        }

    printf ("%d\n", a + b);
    return 0;
    }
```

Стиль **Whitesmith**

Результаты работы

- Предложена модель для анализа стиля форматирования и именования переменных в программе на произвольном языке программирования.
- Исследован вопрос получения информации о синтаксической структуре программы на языке C++.
- Предложен и реализован способ получения дерева разбора с помощью компилятора GCC.
- Реализован прототип, проверяющий стиль форматирования программы на языке C++.

Направления развития

- Улучшение поддержки распространённых стилей (более детальные подтипы лексем)
- Реализация анализатора стиля именования переменных
- Улучшение поддержки исключительных ситуаций
- Поддержка нескольких языков программирования (дополнительные front-end-ы)

Демонстрация