Правила для оформления кода в лабораторных работах по дискретному анализу

- имена локальных и глобальных переменных начинаются с маленькой буквы:

```
int variable = 50;
int main() {
    int i = 10;
    ...
}
```

- имена функций начинаются с большой буквы, все аргументы начинаются с маленькой буквы:

```
void Method(const int variable1, const double variable2);
```

- названия классов и определения типов предваряются префиксом 'T', за которым следует название класса, начинающееся с большой буквы:

```
using TMap = std::map<int, int>;
class TInteger {/*реализация класса*/};
```

- названия абстрактных классов (содержащих чисто виртуальные функции) начинаются с префикса 'I', за которым следует название класса, начинающееся с большой буквы:

```
class IAbstract {
public:
    virtual void Method() = 0;
...
};
```

- названия пространств имен должны начинаться с префикса 'N', за которым следует само название с большой буквы:

```
namespace NAlgorithm {
    ...
}
```

- токены в сложных именах переменных и функций выделяются капитализацией первой буквы токена; локальные и глобальные переменные при этом должны начинаться с маленькой буквы, как было объявлено ранее:

```
void MySuperMethod() {}
class TMySuperClass {
public:
```

```
    void SomeClassMethodWithoutParameters() {...}
...
};

int someGlobalVariable = 50;

int main() {
    int localVariableForIndex = 0;
    ...
}
```

- все глобальные константы должны быть полностью капитализированы:

```
const int FIVE = 5;

int main() {
    ...
}
```

- запрещается использовать директиву define для объявления констант:

```
#define FIVE 5 - запрещено
const int FIVE = 5; - правильно
```

- токены в полностью капитализированных именах констант отделяются символом подчеркивания:

```
const int SOME_LONG_CONSTANT_NAME = 5;
```

- использование символа подчеркивания в качестве начала имени любого объекта запрещено (даже для private-членов класса):

```
class TMyClass {
private:
    int _Data; - запрещено
    void _PrivateMethod() {} - запрещено
    int Data; - правильно
};
```

- использование табуляции в исходном коде запрещено; если нужно вывести знак табуляции, используйте символ '\t':

```
std::cout << someKey << '\t' << someValue << std::endl;
```

- размер отступа равен 4 (четырем) пробелам:

```
int main() {
....int i = 0;
....for (; i < 10; ++i) {
........std::cout << i << std::endl;
....}
....return 0;
}
```

- все блоки должны быть оформлены в стиле 1TBS или K&R-стиле; однострочные блоки обязательно должны обрамляться фигурными скобками:

```
if (count > 100) {
    return false;
}

while (item.IsValid()) {
    item.DoSomeJob();
    item.IncreaseCounter();
}

for (int i = 0; i < 1000; ++i) {
    std::cout << i << std::endl;
}
```

- для условных операторов, которые не помещаются в одну строку, символ начала блока переносится на следующую пустую строку:

```
if (count < 1000 && item.IsValid() && George.IsCowboy()
    && winter.IsComing())
{
...
}
```

- нельзя использовать несколько операторов в одной строке:

```
int a = 5; int b = 6; − запрещено
while (i < 1000) { std::cout << i << std::endl; ++i; } − запрещено
```

- все знаки бинарных операций отбиваются пробелами с обеих сторон (выделено точкой в примерах):

```
if (a.<.b) {...}
int a.=.b;
```

- категорически запрещается использование неименованных констант:

```
if (count > 369) {...} – запрещено
```

Правильно

```
const int SOME_USEFUL_CONSTANT = 369;
...
if (count > SOME_USEFUL_CONSTANT) {...}
```

- запрещается использовать auto в случаях, когда тип явно выводится из сигнатуры используемого метода; в остальных случаях использование auto не запрещается, если при защите работы автор может объяснить, на какой тип следует заменить auto, чтобы код не изменил поведения:

```
int SomeMethodThatReturnsInteger();
std::string SomeMethodThatReturnsString();
```

Запрещено

```
auto x = SomeMethodThatReturnsInteger();
auto y = SomeMethodThatReturnsString();
```

Правильно

```
int x = SomeMethodThatReturnsInteger();
std::string y = SomeMethodThatReturnsString();

std::map<std::string, std::string> data;
for (const auto& item : data) {
    // тип item?
}
```