Функции и файлы
inline, static

Ключевое слово auto

auto – используется для автоматического вывода типа компилятором.

Например:

```
auto var1 = 10L;
auto var2 = 0.2;
auto var3 = 123;
auto var4 = 'x';
```

Директивы препроцессора

`#include` — вставляет текст из указанного файла
`#define` — задаёт макроопределение (макрос) или символическую константу
`#undef` — отменяет предыдущее определение
`#if` — осуществляет условную компиляцию при истинности константного выражения
`#ifdef` — осуществляет условную компиляцию при определённости символической константы
`#ifndef` — осуществляет условную компиляцию при неопределённости символической константы
`#else` — ветка условной компиляции при ложности выражения
`#elif` — ветка условной компиляции, образуемая слиянием else и if
`#endif` — конец ветки условной компиляции
`#line` — препроцессор изменяет номер текущей строки и имя компилируемого файла
`#error` — выдача диагностического сообщения
`#pragma` — действие, зависящее от конкретной реализации компилятора.

Функции

- Синтаксис объявления функции:

  тип_возвращаемого_значения имя функции(тип_формального_параметра имя_формального_параметра, …);

- Синтаксис вызова функции:

  имя_функции(имя_фактического_параметра, …);

```cpp
#include <iostream>
#include <cmath>

double readTriangleSide();
bool isTriangleExists(double leftSide, double rightSide, double bottomSide);
double triangleSemiperimeter(double leftSide, double rightSide, double bottomSide);
double triangleSquare(double leftSide, double rightSide, double bottomSide);
void showSquare(double square);
void showWarningMessage();

int main()
{
    double leftSide = readTriangleSide();
    double rightSide = readTriangleSide();
    double bottomSide = readTriangleSide();

    if (isTriangleExists(leftSide, rightSide, bottomSide))
        showSquare(triangleSquare(leftSide, rightSide, bottomSide));
    else
        showWarningMessage();

    return 0;
}
```

```
double readTriangleSide()
{
    double triangleSide;

    std::cout << "Enter triangle side: ";
    std::cin >> triangleSide;

    return triangleSide;
}

void showSquare(double square)
{
    std::cout << "Triangle square = " << square << std::endl;
}

bool isTriangleExists(double leftSide, double rightSide, double bottomSide)
{
    return leftSide + rightSide > bottomSide &&
    leftSide + bottomSide > rightSide && rightSide + bottomSide > leftSide;
}

double triangleSemiperimeter(double leftSide, double rightSide, double bottomSide)
{
    return (leftSide + rightSide + bottomSide) / 2;
}
```

```
double triangleSquare(double leftSide, double rightSide, double bottomSide)
{
    double p = triangleSemiperimeter(leftSide, rightSide, bottomSide);

    return sqrt(p * (p - leftSide) * (p - rightSide) * (p - bottomSide));
}

void showWarningMessage()
{
    std::cout << "Triangle doesn't exists" << std::endl;
}
```

Глобальные переменные

Объявление:

```
extern int var;
```

Определение:

```
int variable = 3;
```

Недостатки глобальных переменных:

- Побочные эффекты
- Не стандартизирован порядок инициализации

Статические глобальные переменные

- Статическая глобальная переменная – глобальная переменная, которая видна только в пределах одной единицы трансляции

Определение:

```
static int var;
```

Недостатки глобальных переменных:

- Побочные эффекты

Статические локальные переменные

Время жизни статических локальных переменных – от первого вызова функции, до окончания работы программы

```
int callCount()
{
    static int counter = 0;
    counter++;

    return counter;
}
```

Статические функции

Статическая функция доступна в пределах одного модуля и имеет внутреннюю линковку

Файл 1.cpp

```
static void function()
{

}
```

Файл 2.cpp

```
static void function()
{

}
```

Ключевое слово inline

inline – указание компилятору сделать функцию встраиваемой

- В месте вызова inline – функции должно быть известно её определение
- inline функции можно определять в заголовочных файлах
- При линковке из всех inline - функций выбирается только одна
- Все определения одной и той же функции должны быть идентичными