Типы данных

Простые (встроенные)	**Структурированные (пользовательские, создаваемые программистом)**
int (signed, unsigned, short, long) double, float char bool	Массивы, строки Перечисления Структуры, объединения Классы

Перечисления enum

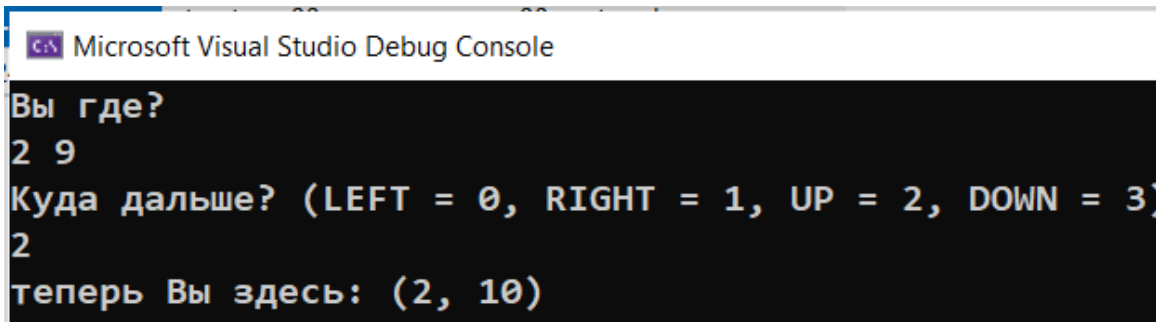
— это определяемый пользователем тип,
состоит из набора
именованных целочисленных констант

```
enum class имя_перечисления {
  константа_1 = значение_1 ,
  константа_2 = значение_2 ,
  ...
  константа_N = значение_N
};
```

**Можно не писать,
но это будет старый С-стиль;
имена констант будут в
глобальной области, возможны
конфликты имен**

Перечисления enum - пример

```
enum class Direction {
    LEFT,  // = 0
    RIGHT, // = 1
    UP,    // = 2
    DOWN   // = 3
};

int main() {
    int x, y;
    cout << "Вы где? \n";   cin >> x >> y;

    Direction dir;
    cout << "Куда дальше? (LEFT = 0, RIGHT = 1, UP = 2, DOWN = 3)\n";
    // cin >> dir;   // нельзя без спец.перегрузки >> для cin

    int tmp;  cin >> tmp;
    dir = static_cast<Direction>(tmp); // (Direction)tmp;

    switch (dir)  {
    case Direction::LEFT: x--;
        break;
    case Direction::RIGHT: x++;
        break;
    case Direction::UP: y++;
        break;
    case Direction::DOWN: y--;
        break;
    default: break;
    }
    cout << "теперь Вы здесь: (" << x << ", " << y << ")";
}
```



Перечисления enum - примеры

```
// явная инициализация
enum class DAYS_IN_MONTHS
{
    January = 31,
    February = 28,
    March = 31,
    April = 30,
    May = 31,
    June = 30,
    July = 31,
    August = 31,
    September = 30,
    October = 31,
    November = 30,
    December = 31
};
```

```
enum class VALUES {
    val1 = 2,
    val2 = 'c',
    val3 = true,
    val4 = -9
};
```

```
enum class Color {
    BLACK,     // = 0
    RED = 5,   // = 5
    BLUE,      // = 6
    GREEN,     // = 7
    DARKRED = RED + 100,   // 105
    DARKBLUE,              // 106
    DARKGREEN,             // 107
    YELLOW = -10,          // -10
    WHITE,
};
```

```
enum class Operation {
    ADD = '+',
    SUBTRACT = '-',
    MULTIPLY = '*',
};
```

Тип структуры - struct

- **Структура struct – составной тип данных, позволяющий объединить под одним именем несколько элементов данных разных типов, а также связать с этими данными функции для их обработки**

Объявление типа структуры (struct)

```
struct имя_типа_структуры {
    тип  элемент_1;
    тип  элемент_2;
   ...
    тип  элемент_N;
} структурные_переменные;
```

- *имя_типа_структуры* – название шаблона структуры. Шаблон структуры описывает новый тип данных, который может быть использован в дальнейшем в программе на C++;
- *тип* – тип данных, которые имеются в наличии на данный момент в программе;
- *элемент1*, *элемент2*, …, *элементN* – названия элементов (переменных), входящих в структуру, их называют полями структуры;
- *структурные_переменные* – это переменные, которые являются структурами. В случае объявления шаблона структуры *структурные_переменные* могут отсутствовать

Структуры – struct, пример

Создать и протестировать структуру для хранения данных об абоненте телефонной компании.
У абонента известны:
- имя
- размер абонентской платы
- тариф поминутной оплаты
- количество минут телефонных звонков

Создание типа данных Abonent

в памяти еще никаких таких данных не размещено, место не занято

```
struct Abonent {
    string name { "noname" };
    double user_fee { 100.0 }; // абонентская плата
    double tariff;  // стоимость 1 минуты
    int minutes;    // количество минут

    void read_abonent(); // ввод данных с клавиатуры
    double total_cost(); // расчет суммы к оплате абонентом
};
```

Поля

Функции (методы)

```
struct Abonent {
    string name { "noname" };
    double user_fee { 100.0 }; // абонентская плата
    double tariff;  // стоимость 1 минуты
    int minutes;    // количество минут

    void read_abonent(); // ввод данных с клавиатуры
    double total_cost(); // расчет суммы к оплате абонентом
};
int main() {
 // …….
}

// расчет суммы к оплате абонентом
void Abonent::read_abonent() {


   cout << "Имя = "; getline(cin, name);
   cout << "Аб.плата = "; cin >> user_fee;
   cout << "Тариф = "; cin >> tariff;
   cout << "Время = "; cin >> minutes;
}

// расчет суммы к оплате абонентом
double Abonent::total_cost() {
    return user_fee + minutes * tariff;
}
```

Создать и протестировать структуру для хранения данных об абоненте телефонной компании.
У абонента известны:
- имя
- размер абонентской платы
- тариф поминутной оплаты
- количество минут телефонных звонков

Определение функций-членов структуры вне ее (после main или в отдельном h-файле)

`Abonent::`

Функции структуры имеют прямой доступ к ее полям.

```
struct Abonent {
    string name { "noname" };
    double user_fee { 100.0 }; // абонентская плата
    double tariff;  // стоимость 1 минуты
    int minutes;    // количество минут
    void read_abonent(); // ввод данных с клавиатуры
    double total_cost(); // расчет суммы к оплате абонентом
};
```

Использование переменных типа структура,

данные уже размещаются в памяти

```
int main() {

  Abonent a{ "Simpson G.", 200.0, 1.5, 120 };
```

Создание и инициализация переменной

```
  cout << a.name

       << " Должен заплатить " << a.total_cost() << endl;
}
```

Обращение к полю структуры

Вызов функции-члена структуры

```
void Abonent::read_abonent() {
  cout << "Имя = "; getline(cin, name);
  cout << "Аб.плата = "; cin >> user_fee;
  cout << "Тариф = "; cin >> tariff;
  cout << "Время = "; cin >> minutes;
}

double Abonent::total_cost() {
  return user_fee + minutes * tariff;
}
```

```
struct Abonent {
    string name { "noname" };
    double user_fee { 100.0 }; // абонентская плата
    double tariff;  // стоимость 1 минуты
    int minutes;    // количество минут
    void read_abonent(); // ввод данных с клавиатуры
    double total_cost(); // расчет суммы к оплате абонентом
};

int main() {

  Abonent a{ .name = "Simpson G.",
             .tariff = 1.5,
             .user_fee = 200.0,
             .minutes = 120 };

  cout << a.name
       << " Должен заплатить " << a.total_cost() << endl;
}

void Abonent::read_abonent() {
  cout << "Имя = "; getline(cin, name);
  cout << "Аб.плата = "; cin >> user_fee;
  cout << "Тариф = "; cin >> tariff;
  cout << "Время = "; cin >> minutes;
}

double Abonent::total_cost() {
  return user_fee + minutes * tariff;
}
```

Инициализация с явным указанием полей;

можно поменять их местами

```
struct Abonent {
    string name { "noname" };
    double user_fee { 100.0 }; // абонентская плата
    double tariff;  // стоимость 1 минуты
    int minutes;    // количество минут
    void read_abonent(); // ввод данных с клавиатуры
    double total_cost(); // расчет суммы к оплате абонентом
};

int main() {

  Abonent a{ "Simpson G.", 200.0, 1.5, 120 };
  cout << a.name
       << " Должен заплатить " << a.total_cost() << endl;


  Abonent b; // по умолчанию name="noname", user_fee = 100.0, tariff=??? minutes=???

  cout << b.name << " Должен заплатить " << fixed <<  b.total_cost() << endl;


  b.read_abonent();
  cout << b.name << " Должен заплатить " << b.total_cost();
}
```

Значения «по умолчанию»

Не инициализированная переменная;
в полях без значений по умолчанию – «мусор»

Ввод данных с клавиатуры в структурную переменную через пользовательскую функцию структуры

```
Microsoft Visual Studio Debug Console
Simpson G. Должен заплатить 380
noname Должен заплатить 795081179890749876507383325032940795605413308811524981190703954400378888.000000
Имя = Bill G.
Аб.плата = 200
Тариф = 45
Время = 100
Bill G. Должен заплатить 4700.000000
```

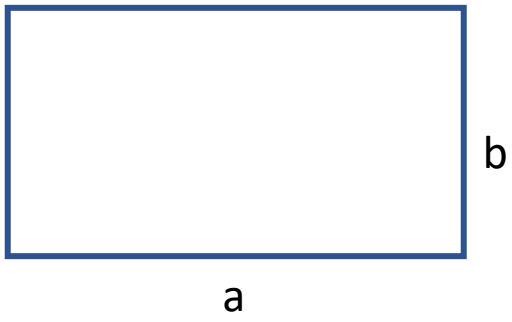
Еще про структуры

- Можно управлять доступом к членам структуры: private, public (default)
- Можно сделать конструкторы – специальные функции для инициализации начальными значениями
- Структура может включать поля, которые сами являются структурами, массивами,…
- Структуры можно наследовать, создавая новые типы на основе уже построенных
- Для динамических переменных типа структура обращение к членам через спец.оператор ->

- Тип Объединение union – аналог структуры с перекрытием полей в памяти

…

Проблема:



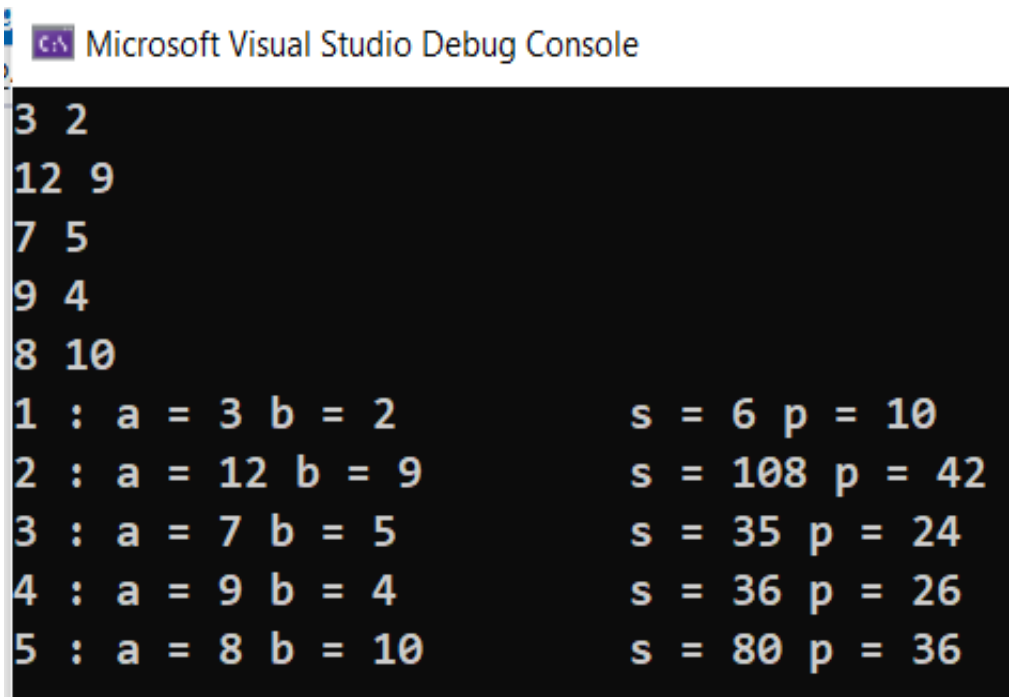

```
double area(double a, double b){
    return a * b;
}

double perimeter(double a, double b){
    return 2.0 *(a + b);
}
```

Написать программу, в которой ввести и разместить в одномерных массивах данные о 5-ти прямоугольниках,
вычислить и вывести их площади и периметры и

- вывести наименьший из периметров,
- вывести количество площадей больших 10,
- вывести сумму площадей,
- средний периметр,
- медианную площадь

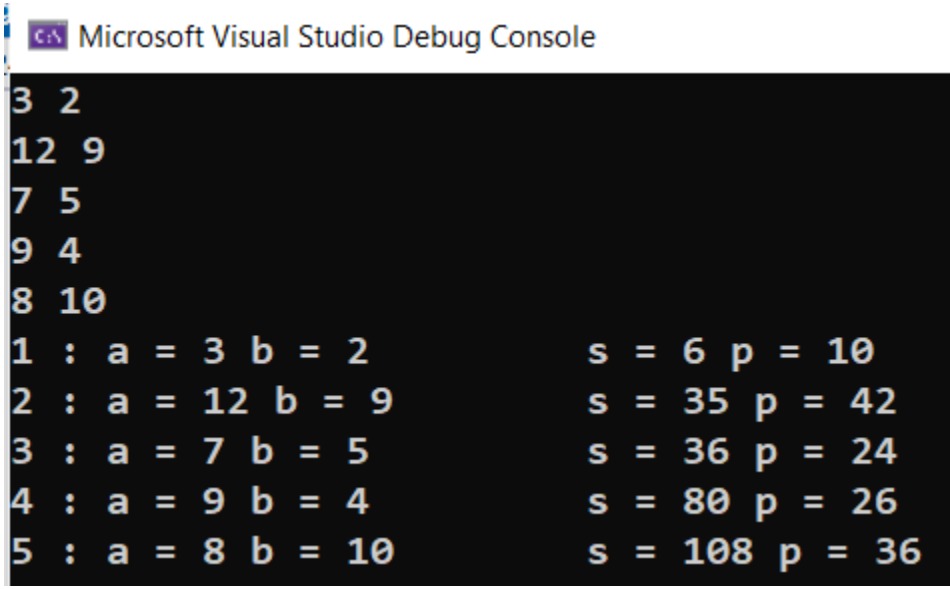
Проблема:

```
double area(double a, double b){
    return a * b;
}

double perimeter(double a, double b){
    return 2.0 *(a + b);
}

int main() {
    const int count = 5;
    double a[count], b[count],
           s[count], p[count];

    for (int i = 0; i < count; i++) {
        cin >> a[i] >> b[i];
        s[i] = area(a[i], b[i]);
        p[i] = perimeter(a[i], b[i]);
    }

    for (int i = 0; i < count; i++) {

        cout << i + 1 << " : a = " << a[i] << " b = " << b[i]
                      << " s = " << s[i] << " p = " << p[i] << endl;
    }
    return 0;
}
```



Проблема:

```
#include <algorithm>

double area(double a, double b){
    return a * b;
}
double perimeter(double a, double b){
    return 2.0 *(a + b);
}

int main() {
    const int count = 5;
    double a[count], b[count],
          s[count], p[count];

    for (int i = 0; i < count; i++) {
        cin >> a[i] >> b[i];
        s[i] = area(a[i], b[i]);
        p[i] = perimeter(a[i], b[i]);
    }

   sort(s, s + count);  // сортировка по возрастанию
    for (int i = 0; i < count; i++) {

        cout << i + 1 << " : a = " << a[i] << " b = " << b[i]
                       << " s = " << s[i] << " p = " << p[i] << endl;
    }
    return 0;
}
```



Проблема:

```
double area(double a, double b){
    return a * b;
}

double perimeter(double a, double b){
    return 2.0 *(a + b);
}

int main() {
    const int count = 5;
    double a[count], b[count],
           s[count], p[count];

    for (int i = 0; i < count; i++) {
        cin >> a[i] >> b[i];
        s[i] = area(81, b[i]);
        p[i] = perimeter(a[i], 25);
    }

    for (int i = 0; i < count; i++) {

        cout << i + 1 << " : a = " << a[i] << " b = " << b[i]
                      << " s = " << s[i] << " p = " << p[i] << endl;
    }
    return 0;
}
```

Проблема:

```
for (int i = 0; i < count; i++) {

    cin >> a[i] >> b[count – i - 1];

    s[i] = area(a[i], b[i]);
    p[i] = perimeter(a[i], b[i]);
}
```

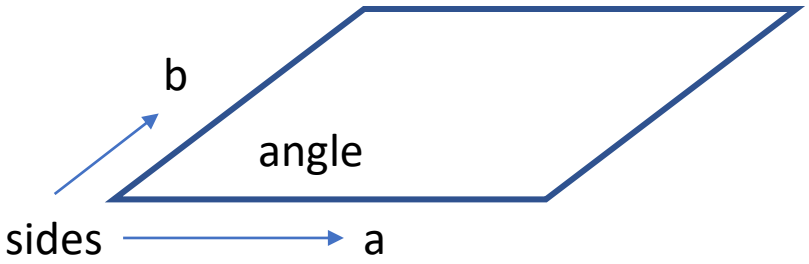
```
for (int i = 0; i < count / 2; i++) {
    cin >> a[i] >> b[i];

    s[i] = area(a[i], b[i + 1]);

    p[i] = perimeter(a[i + 1], 10000000);
}
```

Проблема:

3. Создать функцию, вычисляющую по известным сторонам и углу между ними параллелограмма его площадь и периметр

$S = ab \cdot \sin \gamma$

Написать программу, в которой ввести и разместить в одномерных массивах данные о 10-ти параллелограммах, вычислить их площади и периметры и вывести наименьший из периметров, количество площадей больших 10, сумму периметра первого и площади последнего параллелограмма.



Проблема:

осторожно, это Java

```
double[] sides = new double[20];
double[] angles = new double[10];
for (int i = 0; i < 10; i += 2) {
    System.out.print("side's length " + (i + 1) + ": ");
    sides[i] = scanner.nextDouble();
    System.out.print("agnle between " + (i + 1) + " and " + (i + 2) + ": ");
    angles[i] = scanner.nextDouble();
    System.out.print("side's length " + (i + 2) + ": ");
    sides[i + 1] = scanner.nextDouble();
}
double[] areas = new double[10];
double[] perimeters = new double[10];
for (int i = 0; i < 10; i += 2) {
    areas[i] = sides[i] * sides[i + 1] * Math.sin(angles[i]);
    perimeters[i] = 2 * (sides[i] + sides[i + 1]);
}
```

Решение ?

```
struct Rectangle {
    double a;
    double b;

    void read_reactangle() {
        cout << "side_a = "; cin >> a;
        cout << "side_b = "; cin >> b;
    }

    double area() {
        return a * b;
    }
    double perimeter() {
        return 2.0 * (a + b);
    }
};

int main() {
    const int count = 5;
    Rectangle r[count];
    for (int i = 0; i < count; i++)
    {
        r[i].read_reactangle();
    }

    for (int i = 0; i < count; i++)
    {
        cout << i + 1 << " : a = " << r[i].a << " b = " << r[i].b
            << " \ts = " << r[i].area() << " p = " << r[i].perimeter() << endl;
    }
}
```