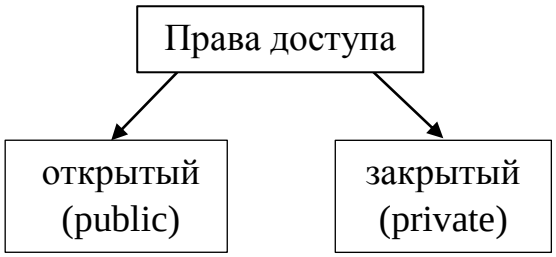
Права доступа

В C++ существует механизм ограничения доступа к полям и методам структурных типов данных. В связи с этим существуют два (на самом деле три, но об этом позже – когда речь пойдет о наследовании) уровня доступа к полям и методам:



Уровень доступа public означает, что соответствующие поля и методы открыты, т. е. они доступны для любой функции программы. Этот уровень доступа используется для тех полей и методов, которые программист предполагает использовать в других частях программы.

Уровень доступа private указывает, что соответствующие поля и методы закрыты, т. е. они доступны только из методов данного структурного типа данных. Данный уровень доступа используется в тех случаях, когда программист планирует использовать соответствующие поля и методы только внутри своего структурного типа данных.

В структуре *по умолчанию* все поля и методы являются *открытыми* (слово public можно не указывать):

Пример 1.

```
struct Time
{
   int hour, min, sec;
   void print ()
   { cout << hour <<":"<< min <<":"<< sec<<endl; }
};

int main()
 {
  Time t;
  t.hour = 9; t.min = 30;  t.sec = 0; // можно обращаться
  t.print ();              // можно обращаться
```

```
  ..
}
```

В данном примере в функции main мы имеем доступ ко всем полям и методу print структуры Time.

В C++ имеется и другое ключевое слово для определения структурных типов данных – это класс (class).

Отличие класса от структуры только в том, что в классе *по умолчанию* все поля и методы являются *закрытыми* (слово private можно не писать):

Пример 2.

```
class Time
{
  int hour, min, sec;
  void print ()
  { cout << hour <<":"<< min <<":"<< sec<<endl; }
};

int main()
{
  Time t;
  t.hour = 9; t.min = 30;  t.sec = 0; // нельзя обращаться
  t.print ();                // нельзя обращаться
  ..
}
```

В примере 2 все поля и метод print класса Time являются закрытыми. Поэтому в функции main к ним доступа нет.

Если нужно, чтобы какие-то поля и методы класса были открытыми, нужно перед ними написать слово public:

Пример 3.

```
class Time
{
int hour, min, sec;
public:
void print ()
{ cout << hour <<":"<< min <<":"<< sec<<endl;
};

int main()
```

```
{
  Time t;
  t.hour = 9; t.min = 30;  t.sec = 0; // нельзя обращаться
  t.print ();                  // можно обращаться
..
}
```

Метод print стал открытым, поэтому к нему есть доступ в функции main.

Пример 4. В классе Complex поля re и im – закрытые, а все остальные методы, включая конструкторы, являются отрытыми:

```
class Complex
{
     double re, im;
public:
     Complex()
     { re = 0; im = 0;}
     Complex(double r, double i)
     { re = r; im = i; }
     void print()
     { cout << re << '+' << im << 'i' << endl;}
     double mod()
     { return sqrt(re*re + im*im);}

}; // конец структуры

int main()
{
     Complex x(2, 5), y(1, 3), z;
     z = x + y;
     z.print();
     return EXIT_SUCCESS;
}
```

Замечание. Уровень доступа public (private) действует на последующие поля и методы, до тех пор, пока явно не встретится ключевое слово private (public).

Какие поля и методы целесообразно делать открытыми, а какие закрытыми?

Для того, чтобы гарантировать целостность данных, поля обычно объявляют закрытыми, чтобы кто угодно не мог записать в эти поля что угодно. Методы же, как правило, объявляют открытыми, чтобы их можно было вызывать.

Иногда *по соображениям эффективности* требуется открыть доступ к *закрытым* полям и методам класса для некоторых внешних функций. Для этого существует понятие *дружбы*. Функция является другом класса, если она имеет доступ ко всем его полям и методам, даже к закрытым.

Все функции, дружественные данному классу должны быть указаны внутри определения этого класса. А, именно, прототипы (заголовки вместе с ;) таких функций помещаются в самом классе. Перед каждым таким заголовком должно стоять ключевое слово friend.

Пример 5.

```
#include<iostream>
#include <cstdlib>
using namespace std;
class Complex
{
     double re, im;
public:
     Complex()
     { re = 0; im = 0;}
     Complex(double r, double i)
     { re = r; im = i; }
     void print()
     { cout << re << '+' << im << 'i' << endl;}
     double mod()
     { return sqrt(re*re + im*im);}

     friend Complex operator+ (Complex x, Complex y);

}; // конец структуры

Complex operator+ (Complex x, Complex y)
{
     Complex z;
     z.re = x.re + y.re;
     z.im = y.im + y.im;
     return z;
```

```
}
int main()
{
     Complex x(2, 5), y(1, 3), z;
     z = x + y;
     z.print();
     return EXIT_SUCCESS;
}
```

Задание. В задачах 1 и 2 замените структуру на класс.