



Структуры данных и алгоритмы

ООП

Дополнительно



Дружественные классы и функции

Дружественные функции

- **Дружественные функции** – это функции, объявленные вне класса, но имеющие доступ к закрытым и защищенным полям данного класса.
 - Дружественная функция объявляется внутри класса с модификатором **friend**.
 - Дружественные функции не являются членами класса, поэтому им не передается указатель **this**.

Правила описания и особенности дружественных функций:

- 1) Дружественная функция объявляется внутри класса, к элементам которого ей нужен доступ, с ключевым словом **friend**.
- 2) В качестве параметра ей должен передаваться объект или ссылка на объект класса, поскольку указатель **this** данной функции не передается.
- 3) дружественная функция может быть обычной функцией или методом другого ранее определенного класса.

Правила описания и особенности дружественных функций:

- 4) На дружественную функцию не распространяется действие спецификатора доступа, место размещения ее объявления в классе безразлично.
- 5) Одна функция может быть дружественной сразу несколькими классами.
- 6) Дружественная функция не наследуется.

Общий вид (синтаксис) объявления дружественной функции следующий:

```
class className
{
public:
className();
// другие конструкторы
```

```
friend returnType friendFunction(<список
параметров>);
```

```
};
```

Пример дружественной функции

```
#include <iostream>
using namespace std;
class myclass {
    int a, b;
public:
    friend int sum (myclass x);
    void set_ab (int i, int j);
};
void myclass:: set_ab(int i, int j)
{
    a = i;
    b = j;
}

int sum (myclass x)
```

Задание 6. Найдите ошибки в коде. Исправьте их. Сколько раз сработает конструктор. Добавьте для подсчета статическое поле. Какой результат работы программы выведется на консоль?

```
#include <iostream>
using namespace std;
class X
{
private: int val;
public: X(int v = 0) { val = v; };
friend int f(X&);
friend int g(X);
int h(X) { cout << "\nfunc h:("
<< x.val << ")\n"; return x.val; };
};
int f(X &x) { cout << "\nfunc f:(«
<< x.val << ")\n"; return x.val; }
int g(X x) { cout << "\nfunc g:("
<< x.val << ")\n"; return x.val; }
```

```
void main()
{
    X w;
    w = 199;
    X x(2), y(33);
    f(x);
    f(1);
    g(y);
    g(89);
    x.h(x);
    x.h(y);
    y.h(107);
}
```

.Задание 7. Определите для класса String
(char *str; int len):

-String(const char* _str)

-void print()

-friend String append(String &str1, String
&str2);

-friend String append(const char* str1, String
&str2);

-friend String append(String &str1, const
char* str2).

Продемонстрируйте работу класса.

Дружественные классы

- Один класс может быть дружественным по отношению к другому
- Дружественный класс и все его функции-члены имеют доступ к закрытым членам, определенным в другом классе
- Дружественные классы редко используются в практических приложениях

Пример дружественного класса

```
class myclass {  
    int a;  
    int b;  
public:  
    myclass (int i, int j) { a = i; b = j; }  
    friend class Min;  
};  
class Min {  
public:  
    int min (myclass x);  
}  
  
int Min:: min(myclass x)  
{
```

Подставляемые функции

- Короткая функция, которая не вызывается, а подставляется в соответствующее место программы
- Перед подставляемой функцией указывают ключевое слово `inline`
- Подставляемые функции широко используются в классах
- Использование подставляемых функций ускоряет работу программы, но увеличивает размер кода
- Подставляемые функции должны быть маленькими
- В некоторых случаях компилятор может проигнорировать подставляемые функции
- Рекурсивные функции не могут быть подставляемыми

Пример подставляемой функции

```
inline int min(int a, int b)
{
    return a < b ? a : b ;
}
int main()
{
    cout << min(10, 20);
    cout << " " << min(99, 88);
    return 0;
}
```

С точки зрения компилятора эта программа выглядит
`#include <iostream>`

Пример подставляемой функции – члена класса

```
class myclass {  
    int a;  
    int b;  
public:  
    void init (int i, int j);  
    void show();  
};  
inline void myclass :: init (int i, int j) { a = i; b = j; }  
inline void myclass :: show() { cout << a << " "  
<< b << "\n"; }  
int main()  
{ myclass x;  
  x.init(10,20)  
  x.show();  
}
```

Определение подставляемой функции внутри класса

```
class myclass {  
    int a;  
    int b;  
public:  
    void init (int i, int j) {  
        a = i;  
        b = j;  
    }  
    void show()  
    {  
        cout <<  
    }  
};  
int main()  
{
```

- Если функция определена внутри класса – она автоматически становится подставляемой
- Ключевое слово `inline` необязательно
- Конструктор и деструктор могут быть подставляемыми либо по умолчанию, если они определены внутри класса, либо явно

- Создайте класс с именем **ship**, который будет содержать данные об учетном номере корабля и координатах его расположения. Разработайте метод, который будет сохранять данные о корабле, вводимые пользователем, и метод, выводящий данные о корабле на экран. Напишите функцию `main()`, создающую 5 объектов класса **ship**, а затем запрашивающую ввод пользователем информации о каждом из кораблей и выводящую на экран всю полученную информацию.
- Для задания номера создайте класс, одно из полей которого хранит «порядковый номер» объекта. Для этого необходимо иметь еще одно поле, в которое будет записываться количество созданных объектов класса.
- Каждый раз при создании нового объекта конструктор может получить значение этого поля и в соответствии с ним назначить объекту индивидуальный порядковый номер. В класс необходимо включить метод, который будет выводить на экран порядковый номер объекта.
- Для хранения координат корабля используйте два поля типа **angle**, включающий три поля: `int` для числа градусов, типа `float` для числа минут и типа `char` для указания направления (N, S, E и W). Объект этого класса может содержать значение, как широты, так и долготы. Создайте метод, позволяющий ввести координату точки, направление, в котором она измеряется и метод, выводящий на экран значение этой координаты, например, `179°59,9' E`. Кроме того, напишите конструктор, принимающий эти три аргумента. Для вывода символа градусов воспользуйтесь символьной константой `'\xF8'`.

Задание на самостоятельную работу

Постановка задачи «Кошелек студента». Владелец кошелька может выполнить следующие действия с кошельком: добавить деньги в кошелек, взять деньги, пересчитать, посмотреть, дать деньги в долг. Источниками пополнения кошелька могут быть родители, также это может быть зарплата или стипендия.

Задание:

- Добавить в разработанные классы задачи «Кошелек студента»:
 - Дружественные функции
 - Подставляемые функции
 - Статические переменные-члены класса

Контрольные вопросы

- В каких случаях целесообразно использовать дружественные функции?
- Какой принцип ООП нарушают дружественные функции?
- Какие преимущества дает использование подставляемой функции?
- Когда полезны статические переменные-члены класса?
- В чем отличие структуры от класса?
- Приведите пример использования структур?