Дружественные и подставляемые функции
Структуры , объединения в С++

Дружественные функции

- Обычная функция, которая имеет доступ к закрытым и защищенным членам класса
- Объявление дружественной функции – прототип функции помещается внутри класса, указав перед ней ключевое слово friend
- Дружественная функция не является членом класса, поэтому при ее вызове не нужно указывать имя объекта и использовать оператор ‘.’

Преимущества и ограничения использования дружественных функций

Преимущества:

- позволяют перегружать некоторые виды операторов
- облегчают создание некоторых функций ввода-вывода
- полезны, когда несколько классов могут содержать члены, тесно связанные с другими частями программы

Ограничения:

- производный класс не наследует дружественные функции
- дружественные функции не могут содержать спецификатор хранения (static, extern)

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)

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

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)
```

Определение подставляемой функции внутри класса

```
class myclass {
  int a;
  int b;
public:
  void init (int i, int j) {
      a = i;
      b = j;
  }
  void sho
  {
      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

Статические члены класса

- Данные (переменные) и методы класса могут быть статическими

Объявление статических переменных:

- Перед объявлением переменной ставится ключевое слово static
- Компилятор создает только один экземпляр статической переменной, который будет использоваться всеми объектами этого класса
- Все статические переменные инициализируются нулем до создания первого объекта класса

Определение статических переменных

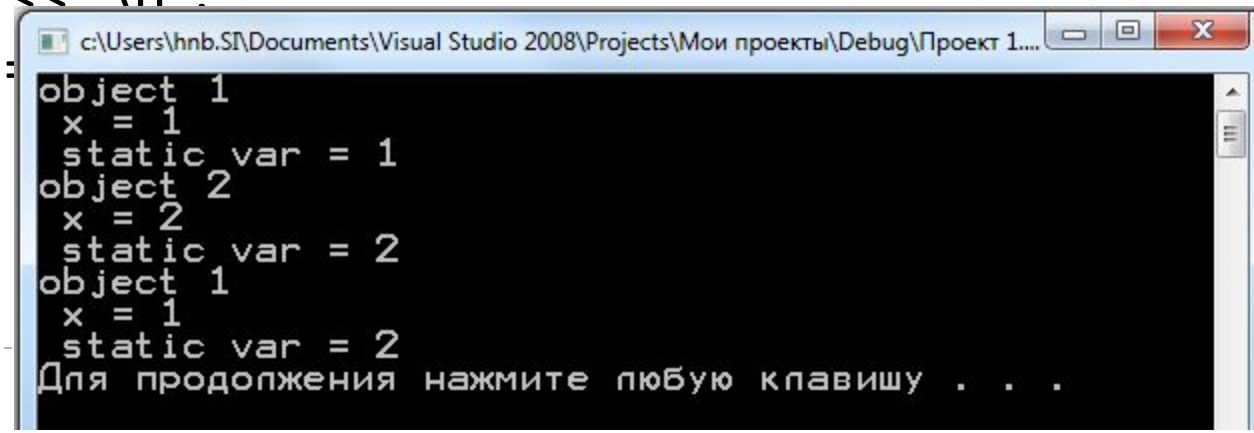
- Объявление статической переменной-члена в классе – это еще не ее определение, то есть память при этом не выделяется
- Статическую переменную необходимо объявить глобально

Тип_переменной Имя_класса ::
имя_стат_переменной;

Пример использования статических переменных

```
class Int {
    int x;
    static int stvar;
public:
    void set(int a, int b) {x=a; stvar=b;}
    void show();
};
int Int::stvar;
void Int::show(){
    cout << " x = " << x << "\n";
    cout << " static var =
};

void main()
{
    Int o1, o2;
```

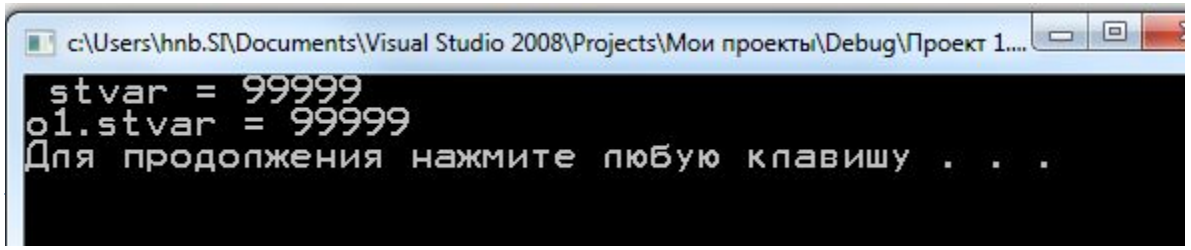


Пример открытой статической переменной

```
class Int {
public:
    static int stvar;
};
int Int::stvar;
void main()
{
    Int::stvar=99999;
    cout << " stvar = " << Int::stvar <<"\n";
    Int o1;
    cout << "o1.stvar = " << o1.stvar << "\n";
    system("pause");
}
```

// переменная открыта и статическая, можем инициализировать до создания объекта

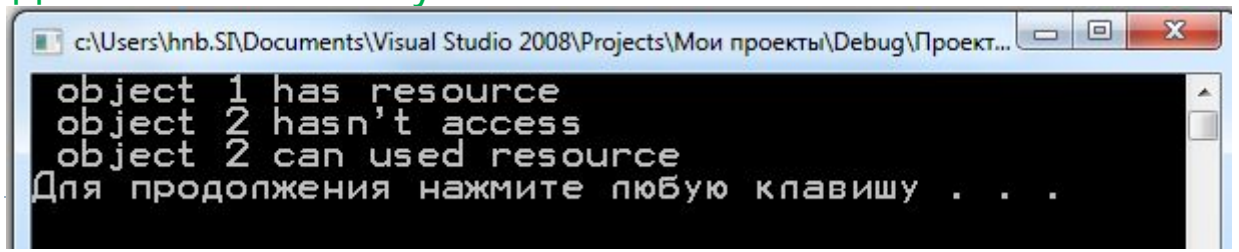
// при обращении к статической переменной независимо от объекта указывается имя класса оператор разрешения области видимости



Управление доступом к ресурсам, которые совместно используются всеми объектами класса, с помощью статических переменных-членов класса

```
class cl {
    static int resource;
public:
    int get_resource();
    void free_resource() { resource = 0;}
};
int cl::resource; // определяем ресурс
int cl::get_resource()
{
    if (resource) return 0; // ресурс занят
    else {
      resource = 1;
      return 1; // ресурс предоставлен объекту
    }
}
```

```
void main()
```



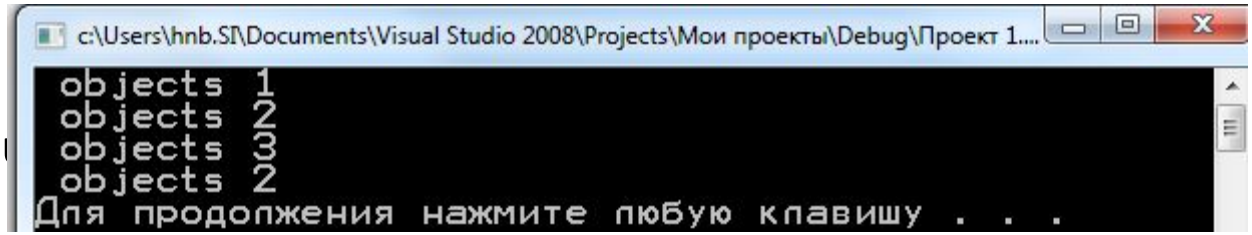
Определение количества существующих объектов класса

```
class Counter {
public:
    static int count;
    Counter () {count ++;}
    ~Counter () {count --;}
};

int Counter::count;

void f();

void main()
{
    Counter o1;
    cout << " objects " << Co
    Counter o2;
    cout << " objects " << Counter::count << "\n";
    f();
```



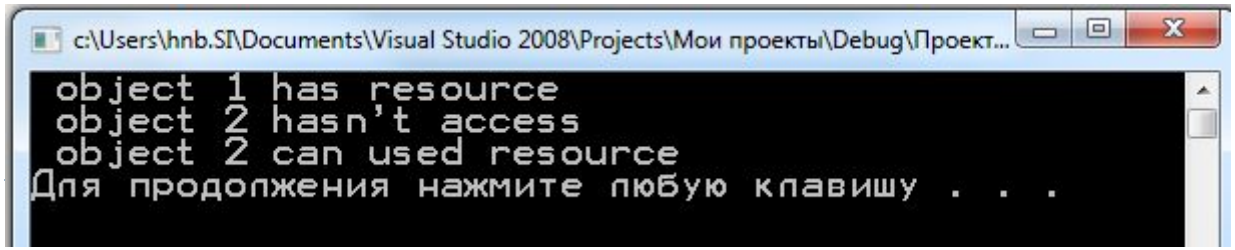
Статические методы класса

- Имеют прямой доступ только к другим статическим членам класса
- Статический метод класса не имеет указатель this
- Одна и та же функция не может иметь одновременно статическую и нестатическую версии
- Статические функции не могут быть виртуальными
- Статические функции нельзя объявлять с помощью ключевых слов const и volatile

Управление доступом к ресурсам с помощью статических переменных-членов и методов класса

```
class cl {
    static int resource;
public:
    static int get_resource();
    void free_resource() { resource = 0;}
};
int cl::resource; // определяем ресурс
int cl::get_resource()
{
    if (resource) return 0; // ресурс занят
    else {
      resource = 1;
      return 1; // ресурс предоставлен объекту
    }
}
```

```
void main()
```



Задание

- 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

Связь между структурами и классами

- Структура – наследие языка С
- Отличие: в языке С++ все члены структуры по умолчанию считаются открытыми, а все члены класса – закрытыми
- В языке С++ структура – разновидность класса

Пример использования структуры вместо класса

```
struct mystr{
void buildstr(char *s); // открытый член
void showstr();
private:
char str[255];
};
void mystr::buildstr(char *s){
if (!*s) *str='\0'; // инициализация строки
else strcat(str,s);
}
void mystr::showstr() {
cout << str << "\n" ;
}
int main(){
mystr s;
s. buildstr("");
s. buildstr("всем");
s. buildstr("привет!");
s. showstr();
```

Связь между объединениями и классами

В языке C++ объединения могут содержать не только данные, но

- функции
- конструкторы и деструкторы
- Объединения могут сохранять все свойства языка C, их члены могут размещаться в одной и той же области памяти
- Отличие: в языке C++ все члены объединения по умолчанию считаются открытыми, и полностью совместимы с языком C
- Ограничения:
- Объединения не могут использовать механизм наследования
- Объединение не может служить базовым классом
- Не может содержать: виртуальные функции; статистические переменные и ссылки; объекты классов, в которых перегружен оператор присваивания; объекты

Безымянные объединения

- Не имеют типа и не могут образовывать объекты
- Сообщают компилятору, что его члены хранятся в одной области памяти
- Доступ к таким переменным осуществляется непосредственно, без помощи оператора ‘.’

```
int main(){
union {
long l;
double d;
};
l=100;
d=12.3;
cout << d<< “”;
return 0;
}
```

Задание

- В вашей модели задачи «Кошелек студента» можно ли использовать структуры?
- Если да, то представьте вариант модели со структурами

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?