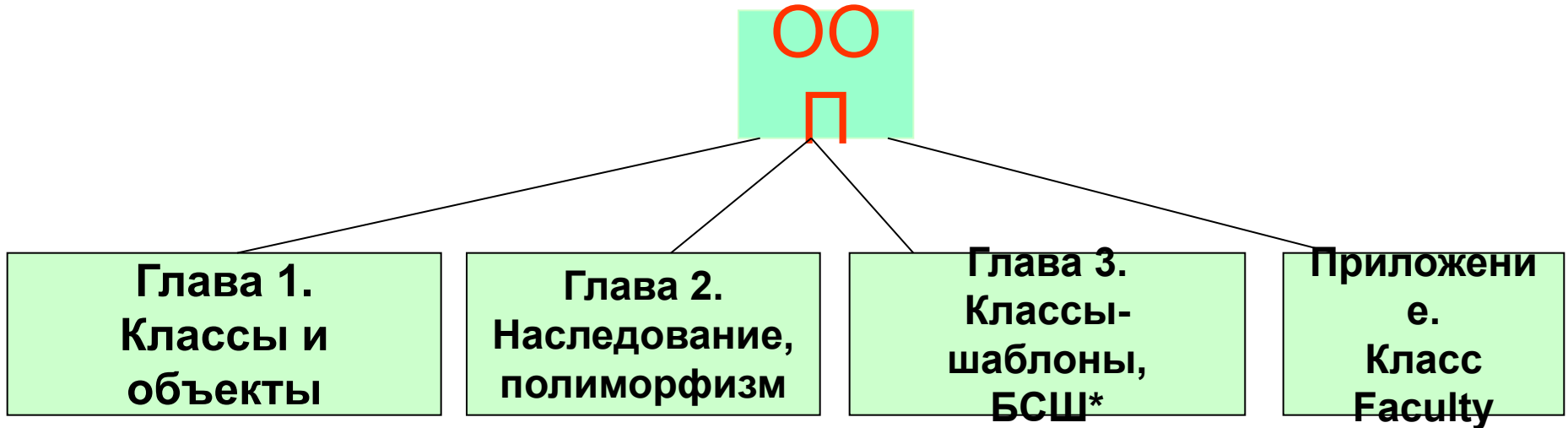


Пахомова Елена Григорьевна

**Объектно-ориентированное
программирование в C++**

ООП

Структура курса



БСШ* – Библиотека Стандартных Шаблонов

Введение. Принципы ООП

ООП – технология разработки **больших** программ

Центральное понятие ООП – **объект**

Объект – это данные и операции(функции), их обрабатывающие, любого уровня сложности.

Причем в ООП **наибольшее внимание** уделяется не реализации объектов, а **связям** между ними

Эти связи организованы в виде **сложных иерархических структур**, где новые типы объектов создаются на основе имеющихся.

Итак,

**Объект = данные + операции и функции,
их обрабатывающие**

В языке C++ имеется большой набор стандартных объектов, но при решении новых задач приходится создавать новые объекты.

ООП базируется на 3-х основных принципах



1. Инкапсуляция - сокрытие информации

Этот принцип предполагает создание пользовательских типов данных, включающих как данные, так и операции и функции, их обрабатывающие.

Никакие другие данные не могут использовать эти операции и функции и наоборот.

Детали взаимодействия данных и функций от пользователя могут быть скрыты.

Контроль за *санкционированным* использованием данных и функций выполняет компилятор

Такие данные называются *абстрактными* в отличие от стандартных (встроенных) типов данных (int, char,...)

Механизм создания абстрактных типов данных осуществляется через понятие *класса*

2. Наследование – создание иерархии абстрактных типов данных

Определяется **базовый класс**, содержащий общие характеристики (прародительский класс).

Из него по правилам наследования строятся **порожденные классы**, сохраняющие свойства и методы базового класса и дополненные своими характерными свойствами и методами.

3. Полиморфизм - множественность форм

Это принцип использования **одинаковых имен функций и знаков операций** для обозначения однотипных действий.

В языке C++ полиморфизм используется в двух видах:

а) для обычных функций и операций над стандартными и абстрактными типами данных.

Это так называемая «**перегрузка функций и операций**»;

б) для функций, определенных в иерархии наследования.

Это так называемые «**виртуальные функции**»

Язык C++ был создан в лаборатории Bell Labs в начале 80-х годов программистом ***Бьярном Страуструпом*** в течение нескольких месяцев путем добавления к C аппарата классов. Первые компиляторы появились в 1985 г.

Литературы много. В НБ:

Буторина Н.Б., Матросова А.Ю., Сибирякова В.А.
**Основы технологии объектно-ориентированного
программирования в языке C++**

Глава 1.

Классы и объекты

1. Определение класса. Скрытие информации

Структура - это комбинированный тип данных, один элемент которого может включать произвольное количество данных *разных* типов, которые называются **полями** структуры.

Формат определения структуры:

```
struct имя_структуры
{тип_поля1 имя_поля1;
  тип_поля2 имя_поля2;
  .....;
};
```

Например,

```
struct anketa
```

```
{char fio[25], fact[10]; int group;};
```

anketa –
новый тип данных

Определение структуры обычно задается вне функций, в начале программы, как глобальное.

Определим переменную

```
anketa p;
```

```
strcpy(p.fio, "Петров");
```

```
strcpy(p.fact, "ФФ");
```

```
p.group = 0546;
```

К полям структуры
обращаемся через точку

▪ - операция выбора

Инициализация

Структуру можно инициализировать при определении переменных

```
anketa s = { "Шарапов", "ФФ", 051761};
```

Можно задать указатель на структуру:

```
anketa *t;
```

```
t = &s;
```

```
t->group = 773;
```

Операция **->** называется «*взять значение поля по адресу*» или «*разадресация*»

-> - сокращение от **(*t)**.

Определение класса базируется на понятии структуры и имеет вид

class имя_класса {тело_класса};

Тело класса содержит определение данных класса – **член-данных**

и объявление или определение функций, их обрабатывающих, – **член-функций**

По иной терминологии ч/данные - **свойства**,
ч/функции - **методы**

Класс String

```
const int MS = 255;  
class String  
{ char line[MS];  
  int len;  
  void Fill(const char *);  
  int Len() { return len;}  
  void Print() { cout << line; }  
  char & Index(int i);  
};
```

объявление
определение
определение
объявление

Здесь *член-данные* - **line, len**;
член-функции - **Fill, Print, Len, Index**.

Член-функции отличаются от обычных функций следующим:

- а) они имеют *привилегированный* доступ к член-данным класса, т.е. используют их непосредственно;
- б) область их видимости(действия) - класс, т.е. они могут использоваться только с переменными этого класса через операцию *'.'*(точка);
- в) член-данные могут располагаться в любом месте описания класса, они *«видны»* всем его член-функциям.

К сожалению,

Таким образом определенный класс мы использовать не сможем.

Единственное, что мы можем – это определить переменные этого типа или указатель

Например,

```
String str1,*str;
```

```
str1.len =10;
```

‘String::len’ is not accessible -

«Переменная **len** из класса String недоступна»

Типы доступа

Для того, чтобы работать с классом, для его член-данных и член-функций надо определить *тип доступа*.

Существует 3 типа доступа:

private член-данные и член-функции доступны **только** член-функциям класса;

protected - член-данные и член-функции доступны член-функциям базового и порожденного классов (гл. 2);

public - член-данные и член-функции общедоступны.

Умолчание

Для классов **по умолчанию** считается доступ – *private*.

Поэтому в нашем примере оказался тип доступа *private* для всех член-данных и член-функций, т.е. всё мы «спрятали в капсулу». (Отсюда термин “*инкапсуляция*”).

Для структур, наоборот, - *public*.

Обычно бóльшую часть член-данных размещают в части *private* - сокрытие информации, а бóльшую часть член-функций – в *public* – интерфейс с программой

Корректируем класс:

```
const int MS = 255;
```

```
class String
```

```
{ char line[MS];
```

```
  int len;
```

```
  public:
```

```
    void Fill(const char *);
```

```
    int Len() { return len;}
```

```
    void Print() { cout << line; }
```

```
    char & Index(int i);
```

```
};
```

Описания *private* и *public* могут стоять в любом месте описания класса и повторяться.

Теперь можно записать оператор

```
int m = str1.Len();    // функция Len() общедоступна
```

Член-функции и операция ::

Вернемся к член-функциям:

две из них определены в классе (Len и Print),
две объявлены (Fill и Index)

Определить объявленные функции можно вне класса, используя операцию '::'

Формат определения:

тип *имя_класса* :: имя_функции (список)
 {тело_функции}

//тип – тип возвращаемого значения

//список – список аргументов

Определим вне класса функции, объявленные в нём:

```
void String:: Fill ( const char *s)
{ for( len = 0; s[len] != '\0'; len++)
    line[len] = s[len];
  line[len]='\0';
}
```

const означает -
s менять нельзя!

```
char & String:: Index( int i )
{ return line[i];
  // функция возвращает i-ый элемент строки
}
```

Вопрос:

Чем отличаются член-функции,
определенные в теле класса и вне
его?

При определении в теле класса функции получают *неявно статус inline*

Поэтому, если функция определена в классе и содержит операторы цикла, то компилятор может выдать *предупреждение* о возможной неэффективности).

Функциям, определенным *вне класса*, также можно присвоить статус *inline явно* первым словом

```
inline char & String:: Index(...){...}
```


2. Объект

Класс - это тип данных, а не объект.

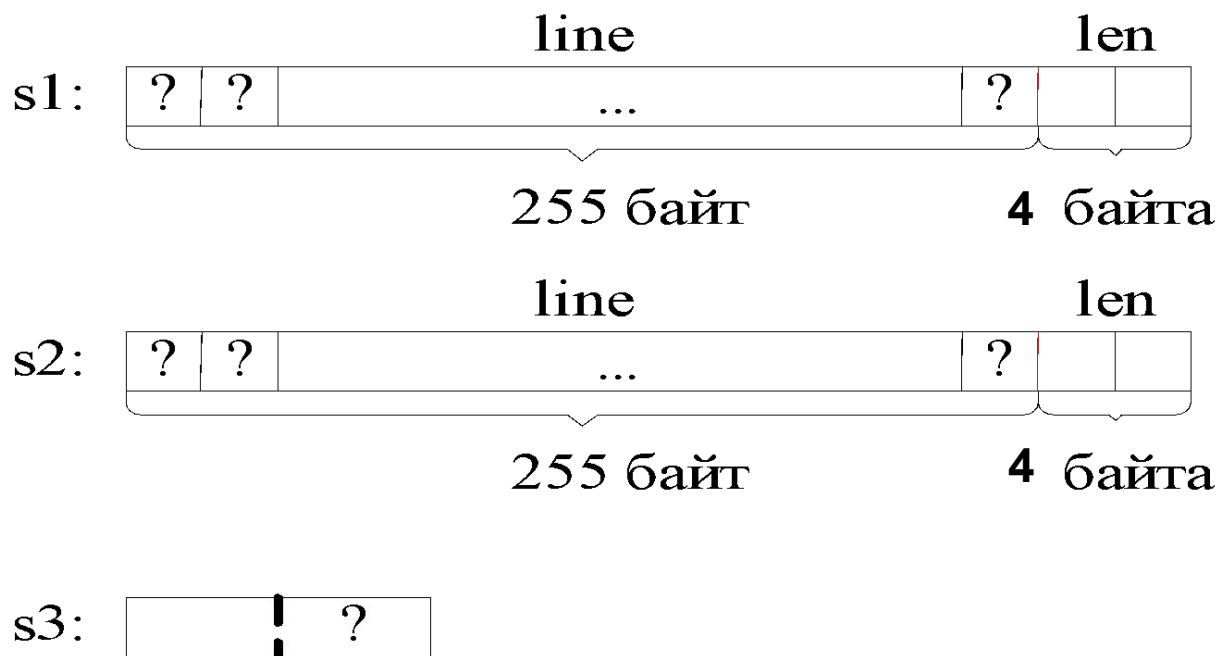
ОПРЕДЕЛЕНИЕ. *Объект* – это переменная, тип которой – класс, и определяется он обычным образом.

```
void main()  
{ String s1, s2, *s3; // s1, s2 - объекты,  
                      // s3 - указатель на объект.  
}
```

Говорят также, что s1, s2 - экземпляры класса.
Для каждого из них будет отведена память по
255 + 4 байтов

Размещение в памяти

? - это мусор



Заметим, что указатель s3 пока не определен, т.е. там тоже мусор.

Работа с объектами

s1.Fill(“объект”);

К ч/функции обращаемся так же,
как к полю структуры (через ‘.’) !

	line						len	
s1:	о	б	ъ	е	к	т \0	...	6

s2.Fill(“ класса String ”);

	line															len		
s2:		к	л	а	с	с	а		S	t	r	i	n	g		\0	...	15

```
void String:: Fill ( const char *s)
{ for(len = 0; s[len] != '\0'; len++) line[len] = s[len] ;
  line[len] != '\0';}
```

Заменим маленькую 'о' на большую в объекте s1

```
s1[0] = 'O';    // ошибка - s1 - это не массив,  
                // и операция [] в нем  
                // не определена!
```

```
s1.line[0] = 'O';
```

```
// ошибка - line - приватное ч/данное,  
// в main (как и в других внешних  
// функциях) его использовать нельзя!
```

s1.Index(0) = 'O';

	line						len	
s1:	Ø	б	ъ	е	к	т \0	...	6

cout << s1.len; // **ошибка:** len – приватное член-данное

cout << s1.Len(); // Так можно получить длину строки

s3 = &s1; // s3 – указатель на строку s1

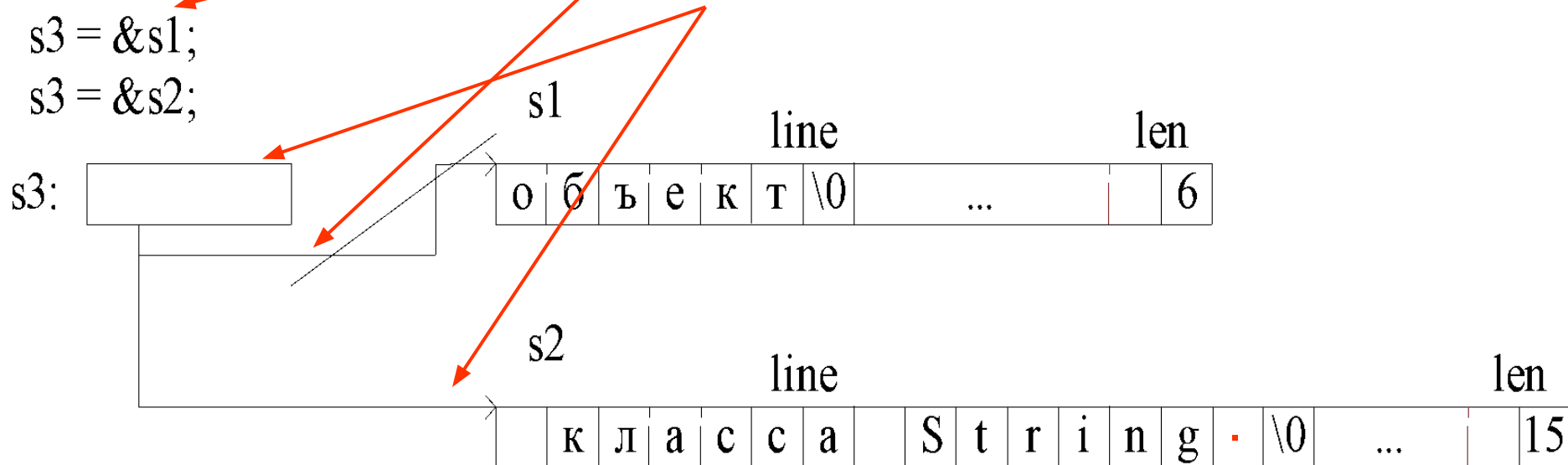
s3 -> Index(0) = 'O'; // Используя функцию Index(int)
// заменим еще раз букву 'O' на 'O'

s3 -> Print(); // Вывод слова «Объект»

`s3 = &s2;` *// теперь s3 - указатель на объект s2*

Эту связь удалили

И связали s3 с объектом s2



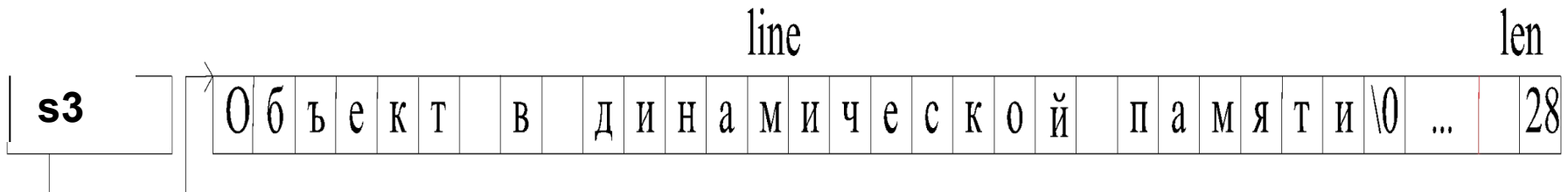
`s3 -> Index(s3->Len ()-1) = '.';` *// Используя член-функции класса*
// Len () и Index() поставим
// в конце строки s2 символ '.'

`s3 -> Print();` *// вывод фразы " класса String."*

Динамический объект

```
s3 = new String;    // Связь с s2 разорвана!  
    // В динамической области(куче) берем память под  
    // поля объекта String.
```

```
s3 -> Fill("Объект в динамической памяти");
```



```
s3 -> Print();  
}
```