1) Полиморфизм – способность функции обрабатывать данные разных типов

1) Полиморфизм; 2) Инкапсуляция; 3) Наследственность;

0. Что такое сигнатура функции?

- тип параметров функции (и их количество)

```
void swap(int* a, int* b);
void swap(double* a, double* b);
```
– разные;

```
int swap(int* a, int* b);
double swap(int* a, int* b);
```
– одинаковые;

2. Что такое (полиморфизм) перегрузка функции в с++? Примеры?

Под перегрузкой функции понимают определение нескольких функций с одинаковым именем, но различной сигнатурой (разн. параметрами)

– избегаем дубли имен; – нет необходимости придум. 10 имен;

– – пример со "swap", также можно изменить кол-во элементов и порядок следования;

Когда выполняют схожие действия с разными типами;

3. Аргументы по умолчанию

Аргумент по умолчанию – это такой аргумент, который программист может не указывать при вызове функции. Он добавляется компилятором автоматически. (и его можно заменить, указав.)

```
void func(int a=val1, int b){ return a*b; }
```

↳ при вызове func(b); будет возвращаться "b*val1

~~4. Что такое тавтология функции?~~

Аргумент должен быть справа (в конце)

Аргумент по умолчанию представляет собой значение, которое используется автоматически, если соответствующий ф. параметр в вызове не указан

(Трата с. 395)

Шаблоны функций

4

(тема 404)

Шаблон функции — это обобщенное описание функции. Передавая шаблону тип в качестве параметра можно заставить компилятор сгенерировать функцию для этого конкретного типа.

Т.к шаблоны позволяют программировать в терминах обобщенного, а не спец. типа, иногда наз. обобщенным программированием. (параметризированные типы).

Покажу, что → это ут. шаблон

```
template <typename Zero>
void swap (Zero &a, Zero &b) {
    Zero t;  t=a;  b=t; }
```

typename ↓ class *

(-) Использование шаблонов не уменьшает используемую память, но сокращает написание кода. В самой программе (хоть мы и не увидим), будут созданы функции, как если бы это было вручную;

Ограничения и специализация шаблонов

Ограничения шаблонов связаны с тем, что не все операции определены для всех типов, например для массива не будет выполняться: a = b;

либо для некоторых структур: if(a > b);

либо для типа указателя: T c = a * b;

⇓

„Явная специализация шаблонов"

Явную специализацию можно задать для отдельных типов. Она будет говорить не генерировать новую функцию под определенный тип, а использовать специализированную;

```
struct job { char name[40]; double salary; int floor; };
```

(*) специализация переопр. шаблон, а нешаб. func переопр. и спец., и шаб.;

далее: ссылки, стр. 371. Прата.

С семинара: &-ссылке (значение может быть присвоено лишь один раз)

Ссылки

6.

Ссылочные переменные (Прата с. 371)

Ссылка представляет собой имя, которое является в c++ псевдонимом или альтернативным именем для ранее объявленной переменной.

Пример: twain – ссылка на переменную clemens, тогда использовать их можно взаимозаменяемо.

① ~~Основное назначение функций~~ :(

Основное назначение ссылок – испол. в кач. формальных аргументов функций. Прим. ссылку в качестве аргумента функция работает с исходными данными, а не с их копиями.

ссылки ~ альтернатива ~ указателям; (при *)

Пример:
```
int rats;
int & rodents = rats; // rodents - ссылка rats;
```

(*) Вводились, прежде всего, для пользовательских типов;

---

7. Зачем нужно ключевое слово inline;

В c++ добавили "inline", теперь функция просто копируется в main без вызова (без call);

Т.е. тело функции копируется прямо на место.

Наиболее выгодно для маленьких функций (иначе – не принцип.)

Иногда компилятор не повинуется, тогда "forceinline".

---

8. auto – определяет значение переменной: auto c = a;

decltype (a+b) c = a+b; – присвоит c тип знач. суммы;