Язык программирования D

Презентация 3.

Стандартная библиотека ввода-вывода. Функции. Владение и передача владения переменных. Рекурсия и замыкания. Управление сложностью кода (пространства имен, модули, пакеты, импорт, экспорт)

Морщинин Н. 5030102/20201

Функции

1. Основная единица структурирования программы
2. Объявляются с указанием типа возвращаемого значения
3. Поддерживают:
    - Значения по умолчанию
    - Перегрузку
    - Вложенные функции
    - Возврат `auto`

Пример простой функции

```
import std.stdio;

int sum(int a, int b) {
    return a + b;
}

void main() {
    writeln(sum(3, 7)); // 10
}
```

Пояснение:

- `int` — тип возвращаемого значения
- `sum` — имя функции
- `return` возвращает результат

Параметры функций

1. `in` — только чтение (константность)
2. `out` — используется для возврата результата
3. `ref` — передача по ссылке (&)
4. `lazy` — вычисление аргумента при обращении

Пример передачи по ссылке

```
void increase(ref int x) {
    x += 5;
}

void main() {
    int n = 10;
    increase(n);
    writeln(n); // 15
}
```

Пояснение:

`ref` позволяет функции изменять исходную переменную.

Возврат значения по ссылке

```
ref int getRef(ref int x) {
    return x;
}

void main() {
    int a = 3;
    getRef(a) = 10;
    writeln(a); // 10
}
```

Комментарий:

Функция возвращает ссылку, а не копию.

Владение и передача владения

1. D позволяет управлять памятью вручную
2. "Владение" — контроль над ресурсом
3. Передача владения — передача ответственности

Пример владения ресурсом

```
struct Resource {
    int* data;

    //Конструктор выделяет память
    this(int n) {
        import core.stdc.stdlib : malloc;
        data = cast(int*)malloc(n * int.sizeof);
    }

    //Деструктор освобождает
    ~this() {
        import core.stdc.stdlib : free;
        if (data !is null) free(data);
    }

    //Копирование запрещено
    @disable this(this);
}
```

Передача владения

```
Resource createResource(int n) {
    return Resource(n);
}

void main() {
    auto r = createResource(5);
}
```

Комментарий:

При возврате структуры из функции владение передается вызывающему коду.

Рекурсия

1. Функция вызывает саму себя
2. Используется для итеративных задач
3. Обязательно должно быть условие выхода

Пример рекурсии

```
ulong factorial(ulong n) {
    if (n <= 1) return 1;
    return n * factorial(n - 1);
}

void main() {
    writeln(factorial(5)); // 120
}
```

Пояснение:

Без базового условия произойдет бесконечный вызов.

Хвостовая рекурсия

```
ulong factTail(ulong n, ulong acc = 1) {
    if (n <= 1) return acc;
    return factTail(n - 1, acc * n);
}

void main() {
    writeln(factTail(6)); // 720
}
```

Комментарий:

Компилятор оптимизирует хвостовую рекурсию до итерации.

Замыкания (Closures)

1. Функция с доступом к внешним переменным
2. В D реализуется через **delegates**
3. Позволяет хранить состояние между вызовами

Пример замыкания

```
int delegate() makeCounter() {
    int count = 0;
    return () {
        return ++count;
    };
}

void main() {
    auto counter = makeCounter();
    writeln(counter()); // 1
    writeln(counter()); // 2
}
```

Пояснение:

`count` сохраняет своё значение между вызовами.

Пример с параметром

```
int delegate(int) adder(int base) {
    return (int x) => x + base;
}

void main() {
    auto add10 = adder(10);
    writeln(add10(5)); // 15
}
```

Комментарий:

Замыкание запоминает значение `base`.

Использование на практике

1. Отложенные вычисления
   (например, передача функции, которая помнит параметры).
2. Создание фабрик функций (как makeCounter).
3. Реализация обработчиков событий и колбэков.

Управление сложностью кода

1. В D вместо пространств имён — **модули**
2. Каждый `.d` файл — это модуль
3. Можно объединять модули в **пакеты** (папки)
4. Импорт выполняется через `import`

Пример модуля

Файл: `math_utils.d`

```
module math_utils;

int square(int x) {
    return x * x;
}
```

Файл: `main.d`

```
import std.stdio;
import math_utils;

void main() {
    writeln(square(4)); // 16
}
```

Пакеты

```
project/
  ├── math/
  │    ├── sum.d
  │    └── mul.d
  └── main.d
```

sum.d

```d
module math.sum;
int add(int a, int b) { return a + b; }
```

main.d

```d
import math.sum;
writeln(add(2, 3)); // 5
```

Импорт и экспорт

- `import` — подключение модуля
- `public import` — экспортирует импорт наружу

Пример:

```
module mylib;
public import std.stdio;

void printMsg(string msg) {
    writeln(msg);
}
```

Использование:

```
import mylib;
printMsg("Hello D!");
```

Практический пример

Задача:

Посчитать сумму квадратов чисел от 1 до 5

```
import std.stdio, std.algorithm, std.range;

int square(int x) { return x * x; }

void main() {
    auto result = iota(1, 6).map!square.sum;
    //iota(1,6) создаёт диапазон
    //map!square применяет функцию
    //sum возвращает результат
    writeln(result); // 55
}
```

Основные модули ввода-вывода

1. `std.stdio` — ввод и вывод в консоль, работа с файлами
2. `std.file` — чтение и запись файлов целиком
3. `std.stream` — потоковый ввод-вывод
4. `std.format` — форматирование строк и данных

Вывод в консоль

```
import std.stdio;

void main() {
    writeln("Hello, D!");          // вывод с переводом строки
    write("Введите имя: ");        // без перевода строки

    string name = readln();        // чтение строки
    writeln("Привет, ", name);     // вывод результата
}
```

- `write` — без новой строки
- `writeln` — с новой строкой
- `readln` — читает строку с консоли

Форматированный вывод

```d
import std.stdio;

void main() {
    int a = 5;
    double b = 3.14;
    writeln("a = ", a, ", b = ", b);

    writefln("a = %d, b = %.2f", a, b); // форматированный вывод
}
```

- `writef` и `writefln` поддерживают **форматные спецификаторы**
- Аналогично `printf` из C, но безопаснее и типобезопасно

Запись в файл

```
import std.stdio;

void main() {
    auto file = File("output.txt", "w"); // открыть для записи
    file.writeln("Строка 1");
    file.writefln("Число: %d", 42);
    file.close();
}
```

- `"w"` — запись (перезапись)
- `"a"` — дозапись в конец файла
- `file.writeln` аналогично `writeln`, но выводит в файл

Чтение из файла

```
import std.stdio;

void main() {
    auto file = File("output.txt", "r");
    foreach (line; file.byLine()) {
        writeln("Прочитано: ", line);
    }
    file.close();
}
```

- `file.byLine()` — безопасный итератор построчного чтения
- Также можно использовать `std.file.readText("имя_файла")` для чтения всего файла

Управление выводом

1. **Буферизация вывода**
   - `stdout.flush();` — сбрасывает буфер вручную
2. **Вывод в другие потоки**
   - `stderr.writeln("Ошибка");`
3. **Перенаправление вывода**
   - можно передать `File` в `writef`, `writeln`, например:

```
auto f = File("log.txt", "w");
writefln(f, "Лог: %s", "запуск программы");
```

Использованные ресурсы

1. https://dlang.ru/book
2. https://habr.com/ru/articles/261043/
3. https://github.com/dlang-community/awesome-d

Спасибо за внимание!