Язык программирования D

Презентация 2.

Типы данных. Операции над типами. Переменные.
Конструкции потока управления.

Морщинин Н. 5030102/20201

Общая информация о типах и переменных

- Строгая типизация
- Начальная инициализация значением по умолчанию
- Название объектов в коде регистрозависимы (`myvar` vs `myVar`)

Типы данных: примитивные

Тип	Определение	Нач. значение
bool	Логическое значение Истина/Ложь	false
byte	8 битное число со знаком	0
ubyte	8 битное число без знака	0
short	16 битное число со знаком	0
ushort	16 битное число без знака	0
int	32 битное число соз знаком	0
uint	32 битное число без знака	0
long	64 битное число со знаком	0

Типы данных: примитивные

Тип	Определение	Нач. значение
ulong	64 битное число без знака	0
float	32 битное действительное число с плавающей точкой	float.nan
double	64 битное действительное число с плавающей точкой	double.nan
real	наибольшое число с плавающей точкой, которое поддерживается оборудованием	real.nan
ifloat	мнимая часть комплексного числа для float	float.nan * 1.0i
idouble	мнимая часть комплексного числа для double	double.nan * 1.0i

Типы данных: примитивные

Тип	Определение	Нач. значение
ireal	мнимая часть комплексного числа для real	real.nan * 1.0i
cfloat	комплексный вариант float	float.nan + float.nan * 1.0i
cdouble	комплексный вариант double	double.nan + double.nan * 1.0i
creal	комплексный вариант real	real.nan + real.nan * 1.0i
char	символ UTF-8 (code point)	0xFF
wchar	символ UTF-16 (code point)	0xFFFF
dchar	символ UTF-32 (code point)	0x0000FFFF

Тип `null`

Основные свойства

- Единственное значение: `null`
- Приводим к любому указателю или ссылочному типу
- Размер: 0 байт (не занимает память)
- Примеры

```
int* ptr = null;          // указатель
Object obj = null;        // класс
string str = null;        // динамический массив
```

Особенности `null`

- `null == null` → `true`
- `typeof(null).stringof` → `"typeof(null)"`
- Не может быть типом переменной (только значение)
  - `null a //ошибка компиляции`

Проверки

```
if (ptr is null)
    writeln("проверка на null");
if (ptr !is null)
    writeln("проверка на не-null");
```

Модификаторы

- const
  - `const int a = 0; // константа этапа компиляции`
- immutable
  - `immutable str a = "immutable str"; // все строки являются immutable`
- shared
  - `shared str a = 0; // для работы с многопоточностью`
- static
  - `static str a = 0; // статическая переменная (на модуль/класс/..)`

Свойства типов (Attributes)

Получение значения через точку `.`

Примеры:

```
import std.stdio;
void main() {
    writeln("Тип                   : ", int.stringof);
    writeln("Длина в байтах        : ", int.sizeof);
    writeln("Минимальное значение  : ", int.min);
    writeln("Максимальное значение: ", int.max);
}
```

Типы данных: дополнение

- Неопределенный тип из пакета `std.variant`
  - `Variant v;`
- Автоматический
  - `auto v = 5;`
- Указатели `type* p`
  - `int* p; // указатель`

Типы данных: дополнение

- Кортежи
  - `auto myTuple = (1, "hello", true);`
- массивы `type []`
  - `int[2] sa; // static array`
- ассоциативные массивы `type [type]`
  - `aa["hello"] = 3;`
- строки
  - `string str = "abcdefg";`
  - `string newstr = str[1..4];`

Типы данных: пользовательские

- *Enum*
  - `enum X { A, B, C }  // named enum, int values`
  - `enum E : char {a, b = char.max}`
- *Struct, Union*
  - `struct S {int i;}`
  - `union U{ubyte i; char c;}`

Типы данных: пользовательские

- *Class*
  - `class Foo { } var; // ERROR`
  - `class Foo { } // no trailing ;`
  - `Foo var;`
  - `class A { }     // A inherits from Object`
- Видимость
  - private, protected, public
- Модификаторы
  - abstract, override, final

- *Interface*

```
interface I {
    void bar() { }  // error, implementation not allowed
    static void foo() { } // ok
    final void abc() { } // ok
}

class A : I1, I2  { } //

class A : I, I  { } // error, duplicate interface

class C : I {
    void bar() { } // ok
    void foo() { } // error, cannot override static I.foo()
    void abc() { } // error, cannot override final I.abc()
}
```

Операции над типами

1. Приведение типов
   - `cast(foo) p; // cast (-p) to type foo`
2. Сравнение типов
   - `is(typeof(i) == const int)`
3. Операции с переменными
   - +-*/
   - str ~ str, [start..end]
   - *ptr

Преобразование в bool

- Базовые типы - `true`, если не ноль
- Указатели - `true`, если не `null`
- Ссылочные типы - `true`, если не `null`
- Пользовательские типы:
  - Вызывается opCast!bool(), если определен
  - Иначе используется alias this, если целевой тип соответствует правилам
  - Перечисления - true, если базовый тип соответствует правилам
  - Если ни одно правило не применимо - ошибка компиляции.

Получение типа

- `typeof ( Expression )`
- `typeof(return)`
- `typeof(this)`
- `typeof(super)`

```
int i;
typeof(i) j; // j is of type int
```

Переменные

1. Объявление (всегда со значением по умолчанию)
    - `int a //a=0`
2. Определение
    - `auto b = 10`
3. Области видимости
    - `{int c = 15}`

Конструкции потока управления: условные конструкции

```
string name = "Maria";
if (name == "Jow")
    sayHello(name); // вызываемая в случае совпадения функция
else if (name == "David")
    sayHello(name); // ...
else
    writeln("Unknow name");
}
```

```
string name = "Maria";
switch(name) {
    case "Jow": // если имя совпало, вызываем функцию, расположенную ниже
        sayHello(name); // вызываемая в случае совпадения функция
        break; // обязательно прерываем выполнение, т.к. искомое значение найдено
    default: // обязательная секция, которая срабатывает, если ни одно из сравнений не совпало
        writeln("Unknow name");
        break;
}
```

Конструкции потока управления: циклы

```
for(int x=0; x<5; x++) {
    writeln(x);
}
```

```
int i = 0;
while(i<5) {
    i++;
    writeln("i: ", i);
}
```

Конструкции потока управления: итераторы

```
import std.stdio;
import std.range; // подключаем модуль работы с итераторами

void main() {
  string [] myarr = ["Jow", "David", "Mike", "Piter", "Maria", "Katerina"];
  foreach(i, element; stride(myarr, 2).enumerate){
    //stride будет возвращать каждый второй элемент
    writefln("index: %s, name: %s", i, element);
  }
}
```

Конструкции потока управления: блоки scope

```
import std.stdio;
void main(){
  writeln("Hello before");
  foo();
  writeln("Hello after");
}

void foo() {
    scope(exit) writeln("1");
    scope(success) writeln("2"); // никогда не будет выведено, т.к. успех не равно ошибке
    scope(exit) writeln("3");
    scope(failure) writeln("4");
    throw new Exception("My Exception"); // кидаем исключение, чтобы сработал блок failure
}
```

Конструкции потока управления: блоки scope

```
Hello before
4
3
1
object.Exception@app.d(): My Exception
```

Использованные ресурсы

1. https://dlang.ru/book
2. https://habr.com/ru/articles/261043/

Спасибо за внимание!