Данный документ является результатом работы компании - avasite, Глотова Жеки, Валуйской Яны, Бротиковской Даны, Ковальчука Саши и Галеева Данилы

си, с++, c#, java, Objective C, Python, JavaScript, Visual Basic, PHP ада83, ада95, Оберон 1/2, Delphi, Modula 1/2, Турбо Паскаль, Lisp, SmallTalk

Prolog, Cobol, Eiffel, Erlang, F#, Groovy, D, Haskel, Perl, Nemerle, Ruby, Scala, Tcl, Ocaml

**C#**

base - ссылка на базовый класс (можно пользовать только при одиночном наследовании) (подобно this)

Структуры нельзя наследовать и от них нельзя наследоваться. Имена структур не являются ссылками - они обозначают непосредственно объект, т.е. они выделяются на стеке. Нельзя определить конструктора по умолчанию

При наследовании в c# в базовом классе для метода нужно писать virtual, а для потомка нужно указывать override, и можно опционально написать sealed, что будет значить - заперт дальнейшей виртуализации функции

Все объекты хранятся в динамической памяти.

where - охрана типа в c# (помимо части LINQ-синтаксиса)

implicit - позволяет пользовать оператор преобразования в c# неявным образом

explicit - только явно (используется по умолчанию)

where T: struct Аргумент типа должен иметь тип значения. Допускается указание любого типа значения, кроме Nullable. Дополнительные сведения см. в разделе Использование допускающих значение NULL типов (Руководство по программированию на C#).
where T : class Аргумент типа должен иметь ссылочный тип; это также распространяется на тип любого класса, интерфейса, делегата или массива.
where T : new() Аргумент типа должен иметь открытый конструктор без параметров. При использовании с другими ограничениями ограничение new() должно устанавливаться последним.
where T : <base class name> Аргумент типа должен являться или быть производным от указанного базового класса.
where T : <interface name> Аргумент типа должен являться или реализовывать указанный интерфейс. Можно установить несколько ограничений интерфейса. Ограничивающий интерфейс также может быть универсальным.
where T : U Аргумент типа, предоставляемый в качестве T, должен совпадать с аргументом, предоставляемым в качестве U, или быть производным от него.

ref, out - передача параметра по ссылке, разница лишь в том, что в out может быть не инициализированным, а в ref - обязательно инициализировано, Их нужно использовать, если в качестве параметра может передаваться не объект (например int), когда передаём объект (да хоть Int32), то тогда передаётся как бы по out.

new - помимо того, что он аллоцирует память, он также может использоваться при виртуальных функциях, чтобы показать, что данная функция не замещает виртуальную функцию базового класса - Важно понимать, что это лишь синтаксический сахар, т.е. это вообще никак не влияет на работу программы. new в данном случае несовместим с override. Так же он может пользоваться при обычных функциях, чтобы если производный класс переопределил функцию базового класса (не виртуальную), то тогда без new - будет warning, а с new - не будет.

event - для объекта делегата добавит ограничение, что событие можно вызвать только изнутри самого класса. Что добавляет безопасности

чтобы работал foreach класс должен реализовывать интерфейс IEnumerable (вообще ещё и Iiterable, но от него наследуется IEnumerable, так что лучше отвечать enumerable)

namespace используется для объявления области видимости

using - имеет 2 смысла - для подключения области видимости и экспортирования оттудова содержимого (не рекурсивно), для создания псевдонимов (для пространств имён или типов), либо для использования как в

using (Font font1 = new Font("Arial", 10.0f))
когда мы создали переменную и её сразу инициализировали. Изменить font1 уже нельзя. А так же корректно вызывается метод Dispose (если мы были унаследованы от IDisposable)

Чтобы в catch поймать абсолютно всё нужно просто написать catch {<заметем, что не было "()">}

Защищённые члены класса (protected, private) отвечают за то, будет ли виден класс из другого пакета, и они не участвуют в ограничениях при наследовании.

//================================================================================================

static implicit operator Y (X x) { ... }

//================================================================================================

// Лямбда функции

Func func1 = x *=>* x + 1;

Func func4 = (int x) *=>* { return x + 1; };

Func func5 = (x, y) *=>* x \* y;

Action func6 = () *=>* Console.WriteLine();

//================================================================================================

Пример стража типа

public class Employee {}

public class GenericList<T> where T : Employee {}

//================================================================================================

class TimePeriod
{
 private double \_seconds;
 public double Seconds
 {
 get { return \_seconds; }
 set { \_seconds = value; }
 }
}

//================================================================================================

Пример явной реализации интерфейса

public class SampleClass : IControl, ISurface
{
 void IControl.Paint()
 {
 System.Console.WriteLine("IControl.Paint");
 }
 void ISurface.Paint()
 {
 System.Console.WriteLine("ISurface.Paint");
 }
}

//================================================================================================

a = new int[3, 5]

int [][] a; /\*первый уровень представляет ссылки на массивы\*/

a = new(int [10][]);

/\*Инициализация\*/

a[0]= new int[10];

a[1]= new int[3];

class Y {...}

struct B{...}

class X {

 Y y; - указатель

 B b; - экземпляр

}

public Student this[int i]

{

 get

 {

 return (Student)InnerList[i];

 }

 set

 {

 InnerList[i]=value;

 }

}

// keywords\_base.cs

// Accessing base class members

using System;

public class Person

{

 protected string ssn = "444-55-6666";

 protected string name = "John L. Malgraine";

 public virtual void GetInfo()

 {

 Console.WriteLine("Name: {0}", name);

 Console.WriteLine("SSN: {0}", ssn);

 }

}

class Employee : Person

{

 public string id = "ABC567EFG";

 public override void GetInfo()

 {

 // Calling the base class GetInfo method:

 base.GetInfo();

 Console.WriteLine("Employee ID: {0}", id);

 }

}

class TestClass

{

 static void Main()

 {

 Employee E = new Employee();

 E.GetInfo();

 }

}

// keywords\_base2.cs

using System;

public class BaseClass

{

 int num;

 public BaseClass()

 {

 Console.WriteLine("in BaseClass()");

 }

 public BaseClass(int i)

 {

 num = i;

 Console.WriteLine("in BaseClass(int i)");

 }

 public int GetNum()

 {

 return num;

 }

}

public class DerivedClass : BaseClass

{

 // This constructor will call BaseClass.BaseClass()

 public DerivedClass() : base()

 {

 }

 // This constructor will call BaseClass.BaseClass(int i)

 public DerivedClass(int i) : base(i)

 {

 }

 static void Main()

 {

 DerivedClass md = new DerivedClass();

 DerivedClass md1 = new DerivedClass(1);

 }

}

//================================================================================================

// Пример реализации сопрограммы

class Program
{
 public static IEnumerable<long> fib()
 {
 yield return 0;
 long i = 0, j = 1;
 while (true)
 {
 yield return j;
 long temp = i;
 i = j; j = temp + j;
 }

 }
 static void Main(string[] args)
 {
 foreach (long n in fib())
 {
 if (n > 1000)
 break;
 Console.WriteLine(n);
 }
 }
}

//================================================================================================

**Java**

super - ссылка на базовый класс (можно пользовать только при одиночном наследовании) (подобно this)

В java - все методы виртуальные

final - слово для метода, которое означает, что его нельзя в дальнейшем наследовать и замещать.

>> - арифметический сдвиг (т.е. с учётом знакового бита (он дублируется)) (в java ведь все типы знаковые (кроме char для символа))

>>> - логический сдвиг (т.е. без учёта знакового бита, т.е. слева всегда появится 0)

чтобы работал for нужно чтобы класс реализовывал Iterable

java для успешной реализации событий, должна унаследоваться от EventObject, EventListener.

package - единица дистрибуции в java - всё, что написано в файле с этим заголовком будет относиться к этому пакету.

Чтобы можно было кинуть исключение, класс должен быть унаследован от Throwable, поэтому если мы перехватываем Throwable, то мы перехватываем всё

Защищённые члены класса (protected, private) отвечают за то, будет ли виден класс из другого пакета, и они не участвуют в ограничениях при наследовании. В то время как в c# - как в с++.

//================================================================================================

// Лямбда функции

n -> n % 2 == 0

n -> n > 3

(арг1, арг2...) -> { тело }

(тип1 арг1, тип2 арг2...) -> { тело }

Runnable r = () -> System.out.println("hello");

//================================================================================================

Пример переопределения типа

class Digit
{
 public Digit(double d) { val = d; }
 public double val;
 // ...other members

 // User-defined conversion from Digit to double
 public static implicit operator double(Digit d)
 {
 return d.val;
 }
 // User-defined conversion from double to Digit
 public static implicit operator Digit(double d)
 {
 return new Digit(d);
 }
}

//================================================================================================

public enum Day {

 SUNDAY, MONDAY, TUESDAY, WEDNESDAY,

 THURSDAY, FRIDAY, SATURDAY

}

или

enum RGBColor {
 public RGBColor(byte R, byte G, byte B)
 private int color;
}
Red(255,0,0)
Green(0,255,0)
Blue(0,0,255)

//================================================================================================

public void open\_file() throws IOException {

 // Statements here that might generate an uncaught java.op.IOException

}

static int [] a;

static {a = new int[10]; for (int i = 0; i < 10; i++) a[i] = 1;}

public class Ff

{

 public int method()

 {

 // ...

 }

}

public class Gg extends Ff

{

 public int method()

 {

 super.method();

 // ...

 }

}

String myMethod()

 {

 try

 {

 return trickyMethod();

 }

 catch ( IOException e )

 {

 return null;

 }

 }

try

 {

 somethingDangerous();

 }

catch ( IOException e )

 {

 System.out.println( "oh oh" );

 throw new BadDataException();

 }

finally

 {

 file.close(); // always executed

 }

//================================================================================================

void f(int N) {
byte [] dynArray = new byte [N];
// ...обработка ...
}

//================================================================================================

class Circle {

public Circle(double radius) {this.radius = radius;}

public double getArea() {return Math.PI\*radius\*radius;}

}

class Cylinder extends Circle {

public Cylinder(double radius, double length) {super(radius);this.length = length;}

public double getArea() { return 2\*super.getArea()+2\*Math.PI\*radius\*length;}

}

//================================================================================================

Пример работающей программы

package Program;

import java.lang.System;
import java.lang.String;

class X
{
 public void g() { System.out.print("1 "); }
 public void f() { g();}
}

class Y extends X
{
 public void g() { System.out.print("2 "); }
 public void f() { g(); }
}

class Z extends Y
{
 public void g() { System.out.print("3 "); }
 public void f() { g(); }
}

public class Program
{
 static X x = new X();
 static Y y = new Y();
 static Z z = new Z();
 public static void main(String[] args)
 {
 Out(); x = y;
 Out(); y = z;
 Out();
 }
 static void Out()
 {
 x.f(); x.g();
 y.f(); y.g();
 System.out.print("\n");
 }
}

Выдаёт

1 1 2 2
2 2 2 2
2 2 3 3
//================================================================================================

**C++**

//================================================================================================

**[** capture **]** **(** params **)** mutable exception attribute **->** ret **{** body **}**

**mutable -** позволяет body изменять параметры, захваченные копированием, и вызывать их неконстантные функции-члены

exception - обеспечивает спецификацию исключения или опцию noexcept для оператора () типа замыкания

attribute - обеспечивает спецификацию атрибута для оператора () типа замыкания

capture - определяет, какие символы, видимые в области, где функция объявлена, будут видны внутри тела функции. Список символов может быть передан следующим образом:

* **[a,&b]** где *a* захвачена по значению и *b* захвачена по ссылке.
* **[this]** захватывает указатель **this** по значению.
* **[&]** захват всех символов по ссылке
* **[=]** захват всех символов по значению
* **[]** ничего не захватывает

//================================================================================================

Вычисление факториала на стадии компиляции

#include <iostream>

template<int n>
class Factorial {
public:
 static const int f = Factorial<n - 1>::f \* n;
};

template<>
class Factorial<0> {
public:
 static const int f = 1;
};

int main() {
 std::cout << Factorial<5>::f << std::endl; // 120
}

//================================================================================================

**Python**

Типы данных бывают -

строки, числа (целые, не целые)

list - список [1,2,3, [4,2,"каждый элемент - это свой тип"]]

dict - словарь {"1" : 23, 34 : ["dd", 23]} - ключём в словаре может случить исключительно hash-able тип, т.е. тип который не изменяем, например - строка или кортеж, ещё бывают frozen\_set, …

Вообще про изменяемось это целая песня

tuple - неизменяемый список, или "кортеж" (1,2,3, (3,2))

Если в функцию передать неизменяемый тип, то любые преобразования над ним в самой функции не будут сохранены, если передать изменяемый - то это как передать по ссылке.

Слова True и False должны писаться с большой буквы

//================================================================================================

my\_func = lambda x, y, z: x+y+x

//================================================================================================

my\_list = [1,2,3,4]

my\_new\_list = [c какого элемента включительно : по какой элемент не включительно : шаг (может быть и отрицателен)]

my\_new\_list = my\_list[3:1:-1] # == [4,3]

my\_new\_list[0] # это элемент 4

В словарях обращение идёт my\_dict[Тут пишем ключ]

//================================================================================================

try

 raise Exception ("Hello World")

except Exception as e:

 print ("hifi") # Такой принт в python 3.0 - в нём все строки unicode

except :

 print "This will catch everything" # Такой принт в python 2.7 - этот с русским не дружит совсем

else:

 print "Это выполнится, если ни один except не случился, т.е. не было исключений"

finally:

 print "Это выполнится вообще всегда"

//================================================================================================

Пример класса

import itertools # Импортирование модуля itertools, чтобы к чему-то достучаться нужно писать "itertools."

from math import sqrt # импортирвоние символа sqrt из модуля math

from math import \* # Это вывалит все имена из модуля math прямо в эту же область видимости (что даст возможность вызвать cos без math.cos)
class Dot (object): # Класс с наследованием (Отдельно заметим, что класс унаследованный из object является классом "нового типа", разницу спросят очень вряд ли)
 def \_\_init\_\_ (self, \*l, \*\*II): # Конструктор, (звёздочки не обязательны), конструкторы базовых типов нужно вызывать вручную
 self.l = list(l)

 def \_\_str\_\_(self): # пример переопределения оператора str(), который возвращает строку
 return ",".join([str(i) for i in self.l])

 def \_\_add\_\_ (self, a2) # пример переопределения оператора "+"

 ...

 def distance(self, d):
 if len(self.l) != len(d.l):
 raise ValueError()
 else:
 return sqrt (sum(map((lambda x1, x2: (x2 - x1)\*\*2), self.l, d.l)))

 def middle(self, d):
 if len(self.l) != len(d.l):
 raise ValueError()
 else:
 return Dot(\*map((lambda x1, x2: ((x1+x2)\*1.0)/2), self.l, d.l))
//================================================================================================

def \_\_find\_max():
 for parameters in [0,1,2,3]: # Второе должно быть чем-то итерабельным, т.е. у него должен быть реализован метод \_\_iter\_\_
 try:
 yield MY\_LAMBDA(\*parameters)
 except:
 pass

//================================================================================================

my\_list = [i\*i for i in range(23)] # Генератор списка - т.е мы по списку range(23) (это специальная функция, которая генерит список [0, 1, 2, 3, …, 22]) генерируем новый список - из квадратов этих чисел.

Аналогично можно и для словаря

range даст нам список, но если нам список не нужен, а нужен только итератор, то нужно пользовать xrange - ибо он негенерирует список и потому не есть памяти

for i in xrange(len(l)):
 if i % 2 == 0:
 if любое условие, без круглых скобок! : # пользуются and or not и ()

# Ещё можно использовать all([список, где для успеха должны быть все True]) и any ([список где хоть один должен быть True])
 print "NO"
 break
else:
 print "YES" - это будет напечатано, если for закончится и при этом не случится break

//================================================================================================

**JavaScript**

//================================================================================================

Функцию можно объявить следующим образом, что является лямбдой

var my\_func = function (int x) {

 return x\*x

};

либо

var my\_func = x => return x\*x;

var my\_func = () => alert(1);

//================================================================================================

Так же существуют генераторы (сопрограммы)

function\*a(){
var b = 0; while(true) { yield b++}
}
a = a();
console.log(a.next().value); // 0
console.log(a.next().value); // 1
console.log(a.next().value); // 2

//================================================================================================

**Ada**

3 смысла with:

 - импортирование модуля (with package; после чего можно иметь доступ к полям по package.field) (эквивалентно #include)

 - Охрана типа (ограничения в шаблонах)

 - with - в случае наследования, в конце добавляет дополнительные поля

use - включение всего содержимого модуля (точнее всего интерфейса), это как в python "from package import \*" (или же это эквивалент using namespace), либо выделение памяти на куче

aliased - тоже умеет размещать данные в пуле

access - указатель, который указывает только на кучу

access all - это указатель, который доступен и для записи и для чтения (может указывать не только на кучу, но и static и стек)

access constant - это указатель, который доступен только для чтения

task - поддерживает многопоточность (можно объявлять как задачи, так и типы задач)

accept - по синтаксису - как поцедура, это сервис, который работа (task) может предоставить кому-то ещё

Если в работе в ходе её выполнения в коде встречается accept - то работа зависает и ждёт, пока кто-нибудь вызовет.

select - (select <accept> or <accept> or … end select) - работа будет ждать не каждого accept по очереди, а лишь хотя бы одного из описанных в select

Разница в in, out и in out в 83 и 95 в том, что out - только для записи в 83, а в 95 (вероятно, не факт что ответ правльный) - мы можем записать в out переменную значение, которое потом можно прочесть оттуда же.

private используется когда описываются "приватные типы" в пакете, "Private child packages" (когда есть слово private перед package - то этот пакет может пользовать только equally private client units), "Private Part in Package Specification" (приватная часть пакета (это целый блок программы package … private … end package))

Package Shelf … - это пакет

Package Shelf.Bool... - это дочерний пакет (он видит всё, что в shelf)

limited - означает, что у новоявленного типа нет стандартных операций до тех пор, пока программист сам их не опишет. (моделировать в java невозможно)

Штрих в ада используется для указания "тип'значение" (например для перечислимого типа, если в разных перечислимых типах используется одна литера (в других языках где литера выпадает в общую область видимости подобное запрещено))

aliased указывает, что к переменной применима адресная операция

//================================================================================================

function F return Some\_Type is

 ... -- declations (1)

begin

 ... -- statements (2)

exception -- handlers start here (3)

 when Name\_1 | Name\_2 => ... -- The named exceptions are handled with these statements

 when others => ... -- any other exceptions (also anonymous ones) are handled here

// others - ключевое слово, перехватывающее всё на свете

end F;

кинуть исключение - raise Name\_1;

//================================================================================================

with persons; use persons;
package managers is

// Наследуемся от person и с дополнительными полями
 type manager is new person with
 record
 master\_key\_code :string(1..10);
 end record;

// Дальше пример метода класса manager
 procedure allocate\_master\_key(
 someone:in out manager;
 code :in string);
 procedure pay\_rise(someone:in out person);
end managers;

//================================================================================================

use Smth\_Package.Entity;
type Entity\_Access is access Entity;
A1, A2 : Entity\_Access;
begin
 A1 := new Entity;
 A1 := new Entity; -- Образовался мусор
 A2 := A1;
 Free\_Entity (A1); -- A1 теперь null
 -- A2 теперь - висячая ссылка
end

//================================================================================================

generic
 type T is private;
 with function “+”(x,y:T) return T (<>);
 with function “\*”(x,y:T) return T (<>);
 type Matrix is private;
 function G\_MatMult(A,B: Matrix) return Matrix;

//================================================================================================

generic
 type T is private;
 Size : integer;
package G\_Queueis
 type Queueis limited private;
 procedure Enqueue(Q: inout Queue; X:T);
 procedure Dequeue(Q: inout Queue; X:T);
 procedure Init(Q: out Queue);
 procedure Destroy(Q: inout Queue);
 function IsFull(Q: Queue);
 function IsEmpty(Q: Queue);
 –- другие процедуры ...
private
 type Queueis record
 Left, Right: integer;
 body : array(1..Size) of T;
 end record;
end G\_Queue;

//================================================================================================

"**Указание типа**"

type RGBCOLOR is (... Red ...);

type TrafficColor is (Red, Green, Blue);

P(RGBColor'Red) - указание использовать тип RGBColor.

//================================================================================================

Двусторонняя связь

procedure Outer is
–- заглушка
procedure Inner is separate;
. . .
end Outer;

separate(Outer)
procedure Inner is
. . .
end Inner

//================================================================================================

function SCAL(X, Y : TARR) return T;

begin

 RES : T;

 RES := 0;

 for i in X'RANGE do loop

 RES := RES+X(i)\*Y(i);

 end loop;

 return RES;

end SCAL;

X:T constant:=e;

package P is

...

type T is ...;

end P;

...

P.T

...

package body P is

type T1 is ...;

end;

//================================================================================================

package Accounts is

 type MONEY is digits 12 delta 0.01;

 type ACCOUNT is tagged private;

 procedure deposit (a: in out ACCOUNT; amount: in MONEY);

 procedure withdraw (a: in out ACCOUNT; amount: in MONEY);

 function balance (a: in ACCOUNT) return MONEY;

 type CHECKING\_ACCOUNT is new ACCOUNT with private;

 function balance (a: in CHECKING\_ACCOUNT) return MONEY;

 type SAVINGS\_ACCOUNT is new ACCOUNT with private;

 procedure compound (a: in out SAVINGS\_ACCOUNT; period: in Positive);

private

 type ACCOUNT is tagged

 record

 initial\_balance: MONEY := 0.0;

 owner: String (1..30);

 end record;

 type CHECKING\_ACCOUNT is new ACCOUNT with null record;

 type SAVINGS\_ACCOUNT is new ACCOUNT with

 record

 rate: Float;

 end record;

end Accounts;

package body REAL\_STACKS is

 procedure put (x: in FLOAT; s: in out REAL\_STACK) is

 begin

 if s.count = s.capacity then

 raise Overflow

 end if;

 s.count := s.count + 1;

 s.implementation (count) := x;

 end put;

 procedure remove (s: in out STACK) is

 ... Реализация remove ...

 end remove;

 function item (s: STACK) return X is

 ... Реализация item ...

 end item;

 function empty (s: STACK) return BOOLEAN is

 ... Реализация empty ...

 end empty;

end REAL\_STACKS;

procedure print\_balance (a: in ACCOUNT'Class) is

begin

 Put (balance (a));

 New\_Line;

end print\_balance;

package Accounts is

 type Account is private;

 procedure Withdraw(An\_Account : in out Account; Amount : in Money);

 procedure Deposit( An\_Account : in out Account; Amount : in Money);

 function Create( Initial\_Balance : Money) return Account;

 function Balance( An\_Account : in Account) return Integer;

private

 type Account is

 record

 Account\_No : Positive;

 Balance : Integer;

 end record;

end Accounts;

with Accounts; use Accounts;

procedure Demo\_Accounts is

 Home\_Account : Account;

 Mortgage : Account;

 This\_Account : Account;

begin

 Mortgage := Accounts.Create(Initial\_Balance => 500.00);

 Withdraw(Home\_Account, 50);

 . . .

 This\_Account := Mortgage;

 if This\_Account = Home\_Account then

 . . .

end Demo\_Accounts;

package Stacks is

 type Stack is private;

 procedure Push(Onto : in out Stack; Item : Integer);

 procedure Pop(From : in out Stack; Item : out Integer);

 function Full(Item : Stack) return Boolean;

 function Empty(Item : Stack) return Boolean;

private

 ...

end Stacks;

package Stacks.More\_Stuff is

 function Peek(Item : Stack) return Integer;

end Stacks.More\_Stuff;

//================================================================================================

**Delphi**

//================================================================================================

Конструкция языка Delphi «uses список\_имен\_модулей» служит для импорта всех имен, объявленных в интерфейсе модулей из списка. При этом импортированные имена становятся непосредственно видимыми (если нет конфликтов с именами из других модулей).

VAR - передача по ссылке, иначе передача по значению

overload в delphi используется после функции для указания, что она будет перегружена (несколько функций с одним именем, но разными параметрами)

//================================================================================================

a: X;
a := X.create; {в этот момент отводится память и вызывается конструктор}
a. – разыменовывание ссылки.

//================================================================================================

// Наследование

TAncestor = class
private
protected
public
 // Виртуальная процедура
 procedure VirtualProcedure; virtual; abstract;
 procedure StaticProcedure;
end;

TDescendant = class(TAncestor)
private
protected
public
 // Перекрытие виртуальной процедуры
 procedure VirtualProcedure; override;
 procedure StaticProcedure;
end;

//================================================================================================

// Пример использования искючений

procedure TForm1.FormCreate(Sender: TObject);
begin
 try
 // Some code which may raise an exception
 except
 on E: EConvertError do
 begin
 // Handle only EConvertError, or any derived classes
 end;
 on E: Exception do
 begin
 // Handle only Exception, or any derived classes
 end;
 end;
end;

//================================================================================================

if ptr = nil then
 raise Exception.Create('Invalid pointer');

//================================================================================================

Двух ступенчатый массив

(На си)

char\*\* mat = new char\*[6];
for (int i = 0; i < 6; i++) mat[i] = new char[i+1];
(На delphi)
var mat: array [0..5] of array of integer;
i : integer;
// <...>
for i := 0 to high(mat) do setlength(mat[i], i + 1);

//================================================================================================

program Overloads; // Название программы
{$APPTYPE CONSOLE} // Фиг знает что
type TRec = record // Описание класса
 private
 function GetOrd: Integer; // Просто функция (она отвечает за свойство, но тем не менее в приватной зоне)
 public
 class operator Implicit(const Value: TRec): Integer; // Это функция для преобразования типов (неявного в частности)
 property ord: Integer read GetOrd; // Описание свойства
 end;
// Ниже пойдёт тела функций класса
class operator TRec.Implicit(const Value: TRec): Integer;
begin
 Result := 0;
end;

function TRec.GetOrd: Integer;
begin
 Result := 0;
end;

procedure Foo(i: Integer);
begin
end;
// Далее обычные объявления
var
 R: TRec;
 a: array[0..0] of Integer;

begin
 Writeln(R); //E2054 Illegal type in Write/Writeln statement
 Writeln(Integer(R)); //explicit cast, provided by class operator Implicit
 Writeln(R.ord); //my preferred option, a property
 a[R] := 0; //E2010 Incompatible types: 'Integer' and 'TRec'
 a[Integer(R)] := 0; //again, explicit cast is fine
 a[R.ord] := 0; //or using a property
 Foo(R); //implicit cast used for actual parameters
end.

//================================================================================================

**Oberon-?**

В Обероне звёздочка означает public, если нету звёздочки - то private ("Скрытый тип данных" - пример ниже).

//================================================================================================

Тут OpaqueT - тип, который экспортируется

TYPE OpaqueT\* = RECORD I:TT; J: TTT END;

//================================================================================================

TYPE T\* = RECORD

Y:T1; // приватно

Z\*:T2; // публично

END;

TYPE T\* = RECORD

 I\* : INTEGER;

 J : INTEGER;

 END

TYPE T1\* = RECORD(T1)

 K : INTEGER;

 END

PROCEDURE (VAR X:T) P(L:INTEGER);

PROCEDURE (VAR X:T1) P(L:INTEGER);

//================================================================================================

**Oberon-2**

В Обероне звёздочка означает public, если нету звёздочки - то private ("Скрытый тип данных" - пример ниже).

//================================================================================================

TYPE T\* = RECORD I\*, J : INTEGER; END;
TYPE T1\* = RECORD (T) K : INTEGER; END; // Тип T - тут базовый класс
PROCEDURE (VAR X: T) P\* (L : INTEGER); // Тут перед названием указывается приёмник

PROCEDURE (VAR X: T1) P\* (L : INTEGER); // Тут перед названием указывается приёмник

//================================================================================================

**Modula-2**

если перед переменой в функции стоит слово VAR - то передача идёт по ссылке, а если без, то по значению

//================================================================================================

DEFINITION MODULE M

TYPE T = ...;

TYPE Stack;

PROCEDURE Init(VAR S: Stack);

PROCEDURE Destroy(VAR S: Stack);

...

END M.

IMPLEMENTATION MODULE M;

...

END M.

//================================================================================================

DEFINITION MODULE HashTables;
 FROM Types IMPORT KeyType, ElementType;
 TYPE HashTable; // Тут объявен скрыты тип данных
 PROCEDURE Init(VAR T:HashTable);
 PROCEDURE Destroy(VAR T:HashTable);
 PROCEDURE Lookup(VAR T:HashTable; Key: KeyType; VAR X:ElementType):BOOLEAN;
 PROCEDURE Add(VAR T:HashTable; Key: KeyType; X:ElementType);
 PROCEDURE Remove(VAR T:HashTable; Key: KeyType):BOOLEAN;
 VAR Done: BOOLEAN;
END HashTables.

//================================================================================================

CLASS Viewers =
 RECORD
 x, y, w, h: INTEGER;
 METHOD restore (T: Text)
 BEGIN ...
 END restore
 END;

 TYPE Viewers =
 RECORD x, y, w, h: INTEGER;
 restore: PROCEDURE (T: Text)
 END;

 v := Viewers.New (X,Y,W,H); NEW (v);
 v.x := X; v.y := Y; v.w := W; v.h := H; v.restore := Restore;

//================================================================================================

**Lisp**

//================================================================================================

[1]> (car '(1 2 3))

1

[2]>

(car не создает список c заголовным элементом.)

//================================================================================================

**Паскаль**

//================================================================================================

type PIneger = ^Integer; // Создание типа

var X: ^Integer: Y: PInteger;

AnotherAddress^ := 99; // Разыменовали оператор

SomeAddress := @SomeNumber; // Взять адрес

New(IntPointer); // Выделить память указателю

**Всякие разные языки**

//================================================================================================

Algol, Ada, Delphi, Modula-2, Pascal - в них понятие

conversion - преобразование типов с какими-либо потерями точности при акрглении, расширении с 32-бит до 64-х, …

casting - любое преобразование типа, в не зависимости от того, это лишь смена интерпретации ("Взглянем на эти же байтики, но немного по другому" (например reinterpret\_cast)), или же это настоящее преобразование типа, с вызовом функции.