```
       C      C++     C#       Java   Oberon        Modula Pascal(16) Delphi    Ada
1s     char           sbyte    byte   BYTE                 shortint   shortint
1u     u. char        byte     -      -                    byte       byte
2s     short          short    short  SHORTINT             integer    smallint
2u     u. short       ushort   -      -                    word       word
4s     int            int      int    INTEGER              longint    integer
4u     u. int         uint     -      -                    ?          cardinal
8s     long long      long     long   LONGINT              -          Int64
8u     u. long long   ulong    -      -                    -          -
arr    int x[N];      ?        T[] x; ARRAY N of T         array[D] of T;       Type ARR is array (IndexT range L..R) of  T;
arr[]  int x[];       ?        ^      ?                    array of T;          Type ARR is array (IndexT range <>)  of T;
!=     !=             !=       !=     ?             #      <>         <>        /=
*T     *T             -        -      ?             ?      ^T         ^T        access T
```

(Ada)

```
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;
```

(C#)

```
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);
    }
}
```

(Java)

```
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
   }

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;}
}
```

(Oberon)

```
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);

(Modula-2)
DEFINITION MODULE M
TYPE T = ...;
TYPE Stack;
PROCEDURE Init(VAR S: Stack);
PROCEDURE Destroy(VAR S: Stack);
...
END M.
IMPLEMENTATION MODULE M;
...
END M.
```

(Ada 95)

```
generic
    type G is private;
package STACKS is
    ... Все, как и ранее, заменяя все вхождения FLOAT на G ...
end STACKS;

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;
```