Объекты и типы

Именование – средство повышения абстракции: использование имени объекта вместо него самого позволяет отвлечься от деталей его реализации.

Области видимости

- *Блочная структура* – иерархия областей, содержащих определения объектов.
- *Правило видимости*: объект, определённый в некоторой области виден в ней самой и всех вложенных областях за исключение тех, где определён одноимённый объект.
- *Однозначность*: в одном блоке не может быть несколько определений одного и того же имени.
- *Поиск определения*: определение для использования объекта с именем X находится в ближайшем охватывающем блоке, содержащем определение объекта с именем X.

Блоки

Пример:

```
int power(float x, int n)
{
    int s = 0;
    for (int k = 0; k <n; k++)
    {
        int ss = s;
        s *= x;
        printf("%f * %f = %f\n", ss, x, s);
    }
}
```

Области видимости

- *Присоединяющий оператор* **with S**: блок, определяющий множество имён из структуры S

  with S do R.X = X

- *Квалификация* – указание охватывающей структуры, типа или библиотеки перед использованием имени

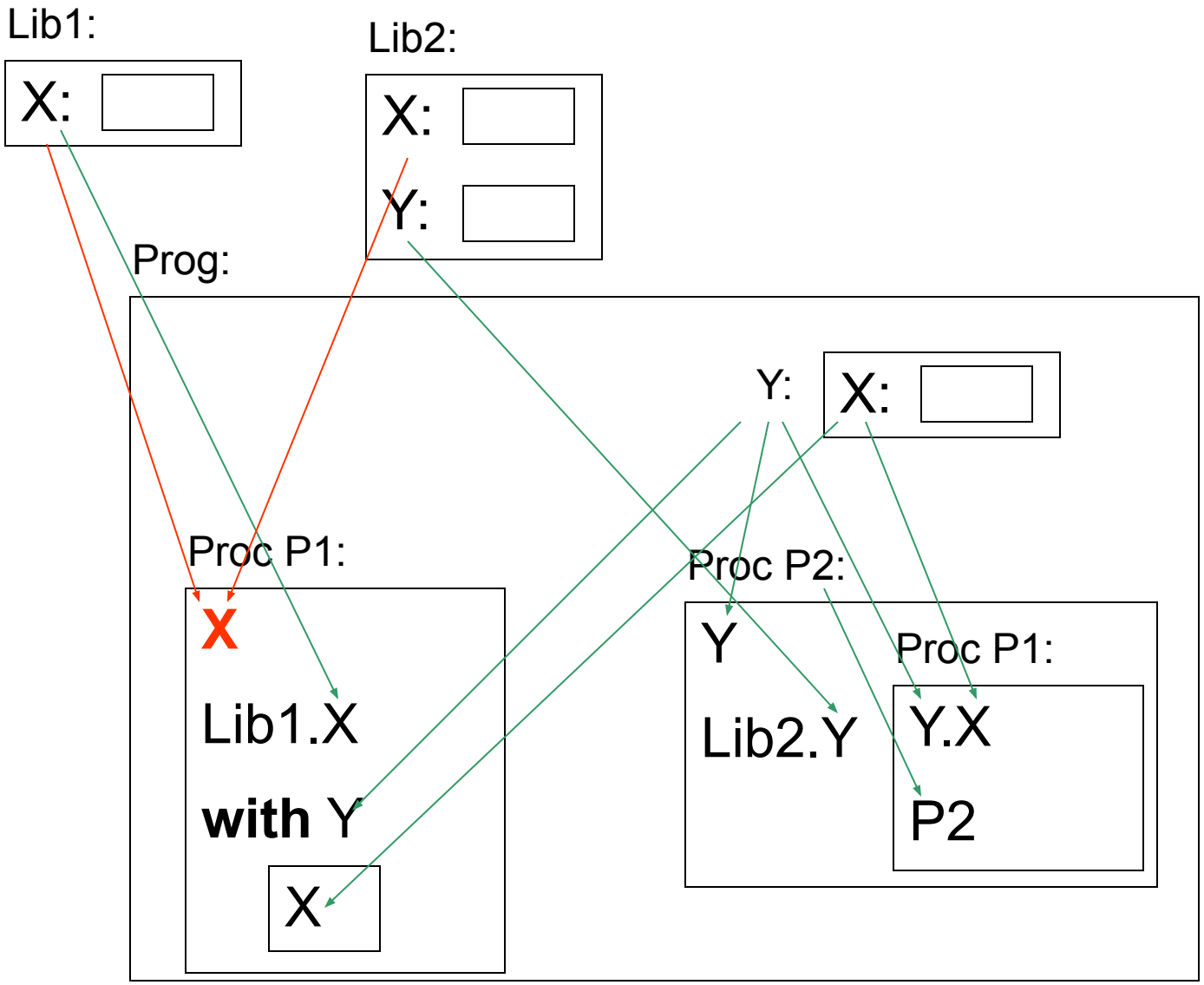
  R.X A[i]->X System.Drawing.Color.Aquamarine

Области видимости - исключения

- *Библиотеки*
  - Конфликты возникают только в момент использования имени
  - Квалификация имён: явное и неявное использование библиотеки
- *Раздельные пространства* имён для разных сортов объектов (SQL):

  select *select*.select **from** *select*, *where* **where** *where*.select=*select*.where;

Области видимости



Анонимные объекты

Объекты, не имеющие собственного имени, доступ к которым осуществляется только через имена других объектов - вычисление имени (адреса).

- Массивы: M[(int) sqrt(R) - 1]
- Указатели: *p, *(p->x[i].y->z)

Типы данных

- Моделируемая категория (например, неотрицательные целые числа)
- Синтаксис (например, unsigned int)
- Литеральные значения – запись констант в тексте программы (например, 0x123)
- Набор операций (например, +, -, *)
- Реализация (например, машинное слово)

Анализ типов

- Статический – тип всех выражений можно выполнить во время трансляции, до исполнения программы
  - Надёжность
  - Понимаемость
- Динамический – тип выражений определяется во время исполнения программы
  - Гибкость (?)
  - Необходим, если новые типы появляются в процессе выполнения

Статический анализ типов

Строгая типизация

- Для каждой переменной, параметра, поля и т. п. указан тип
- Для операций, функций, процедур и т.п. указаны типы аргументов и результатов.
- Разноимённые типы различны (?):

```
typedef int Apples; /* количество */
typedef int Distance; /* в километрах */
typedef int LocalDistance; /* в метрах */
```

Динамическая типизация

Пример:

```
Input x
If x > 0
 y = 2
Else
 y = “2”
End If
Print x + y
```

- Введено “5"
  - напечатано “7”
- Введено “-1"
  - напечатано “-12”
- Введено “Привет!"
  - ошибка при “If x > 0”

Полиморфизм

- *Перегрузка операций*: разные реализации в зависимости от типов аргументов и результатов. Например,
  - 1 + 2, 1.2 + 3.4, “Hello” + “2”
  - void DrawRectangle(int x, int y, int w, int h);
    void DrawRectangle(Location p, Size s);
    void DrawRectangle(Rectangle r);

Полиморфизм

- Родовые типы – типы, имеющие параметры, в том числе и типовые.
  - Реализация операций зависит только от свойств параметров-типов

```
class SortedList <KeyType,ElementType>
{
    public void Insert(KeyType k, ElementType e)
    {  …    }
}
```

  - Пример (C#):

```
SortedList <string,Person> Persons;
SortedList <float,Matrix<float>> Matrices;
Persons.Insert(“Mike”, Me);
Matrices.Insert(A.Determinant(), A);
```

Типы данных

Классификация

- Предопределённые – предоставляемые языком
- Определяемые – описанные в программе

- Простые – неделимые с точки зрения языка, не имеющие компонент
- Структурированные – предназначенные для агрегации компонентов или связи между данными

- Неупорядоченные – присваивание, сравнение на равенство и неравенство: **=**, **==** и **!=** (C), **:=**, **=** и **<>** (Pascal).
- Упорядоченные – кроме того, сравнения **<**, **<=**, **>**, **>=**
- Перечислимые (интегральные) – сопоставление целым
- Арифметические – кроме того, сравнения **+**, **-**, *, /

Логические (Pascal)

Категория	Логические (булевские) значения
Синтаксис	**boolean**
Константы	**true**, **false**
Операции	**and**, **or**, **xor**, **not**
Реализация	байт (слово) : 0 – false, 1 - true

Символы (Pascal)

Категория	Множество символов с кодами ASCII (0..255)
Синтаксис	**char**
Константы	**‘A’**, **‘1’**, **‘*’**, **‘’**,
Операции	chr(‘A’)=64, ord(64)=‘A’
Реализация	байт – двоичное представление ord(c)

256 символов – много или мало?

- 10 цифр + 26 букв + ().,;+-*/ - достаточно
- С другой стороны
  - 32 управляющие кода: перевод строки, возврат каретки, перевод страницы,гудок, табуляция,...
  - 32 символа: пробел, 0..9, …
  - 32 буквы: ABC…XYZ + @[\]^_
  - 32 строчные буквы: abc..xyz + `{|}~…
  - 64 кириллица: АБВ...ЭЮЯабв...эюя
  - 64 псевдографика: ╬╪┘ ...
  - Буква ё, диакритика, лигатуры: ùÿÜ…
  - Греческие: αßπΣГ...
  - Катакана, арабский, иврит, санскрит, иероглифы:...

Многобайтные кодировки

- Shift-JIS – специально для японского
  - Двуязыковая кодировка
  - «Обычные» символы – одним байтом
  - Shift-In, Shift-Out – «скобки» двухбайтовой кодироки
- Universal Character Set (Unicode)
  - Многоязыковые тексты
  - около 100,000 абстрактных символов
- Кодировки
  - UCS-2 – два байта на каждый символ
  - UCS-4 – четыре байта на каждый символ
  - UTF-8 – от одного до четырёх байтов

Целые числа (Pascal)

Категория	Диапазон целых чисел: -32768 .. 32767
Синтаксис	**integer**
Константы	1, -2, 123
Операции	+, *, -, **div**, …
Реализация	не особенно важно

Множества (Pascal)

Категория	Множество всех подмножеств базового типа *T*
Синтаксис	**set of** *T*
Константы	[], [1,2], [‘a’..’z’]
Операции	**+**, *, **-**, **in**, **<=**, **…**
Реализация	Характеристический вектор, битовая шкала

Перечисления (Pascal)

Категория	Набор именованных значений, например, red, green, blue
Синтаксис	**(** *идент* (**,** *идент*)* **)**, например (red, green, blue)
Константы	*идент,* например, red
Операции	упорядоченный
Реализация	отрезок целого типа, например red = 0, green = 1, blue = 2

Целые – представление 1

Неотрицательные

- $b_{n-1}\ b_{n-2}\ \ldots\ b_0$ – последовательность битов
- n – разрядность

$$\sum_{i=0}^{n-1} b_i * 2^i$$

- Диапазон: $0..2^n-1$

Целые – представление 1 (пример)

n=8

- 00000000 = 0
- 00111110 = 32+16+8+4+2 = 62
- 10000100 = 128+4=132
- 11111111 = 128+64+32+16+8+4+2+1 = 255

Целые – представление 2

Со знаком - дополнительный код

- $b_{n-1}\ b_{n-2} \ldots b_0$ – последовательность битов
- n – разрядность.
- b_{n-1} - знак (1 – отрицательные)
- Неотрицательные: $\sum_{i=0}^{n-2} b_i * 2^i$
- Неположительные: $-\sum_{i=0}^{n-2} (1 - b_i) * 2^i$
- Диапазон: $-2^{n-1}-1 \ldots 2^{n-1}-1$

Целые - представление 2 (пример)

n=8

- 0 0000000 = 0
- 0 0111110 = 32+16+8+4+2 = 62
- 1 0000100 = -(64+32+16+8+2+1)=-123
- 1 1111111 = 0

Целые – представление 3

Со знаком - двойное дополнение

- $b_{n-1}\ b_{n-2} \ldots b_0$ – последовательность битов
- n – разрядность

$$-b_{n-1} * 2^{n-1} + \sum_{i=0}^{n-2} b_i * 2^i$$

- Диапазон: -2^n .. 2^n-1

Целые - представление 3 (пример)

n=8

- 0 0000000 = 0
- 0 0111110 = 32+16+8+4+2 = 62
- 1 0000100 = -128+4 = -124
- 1 1111111 = -128+64+32+16+8+4+2+1 =-1

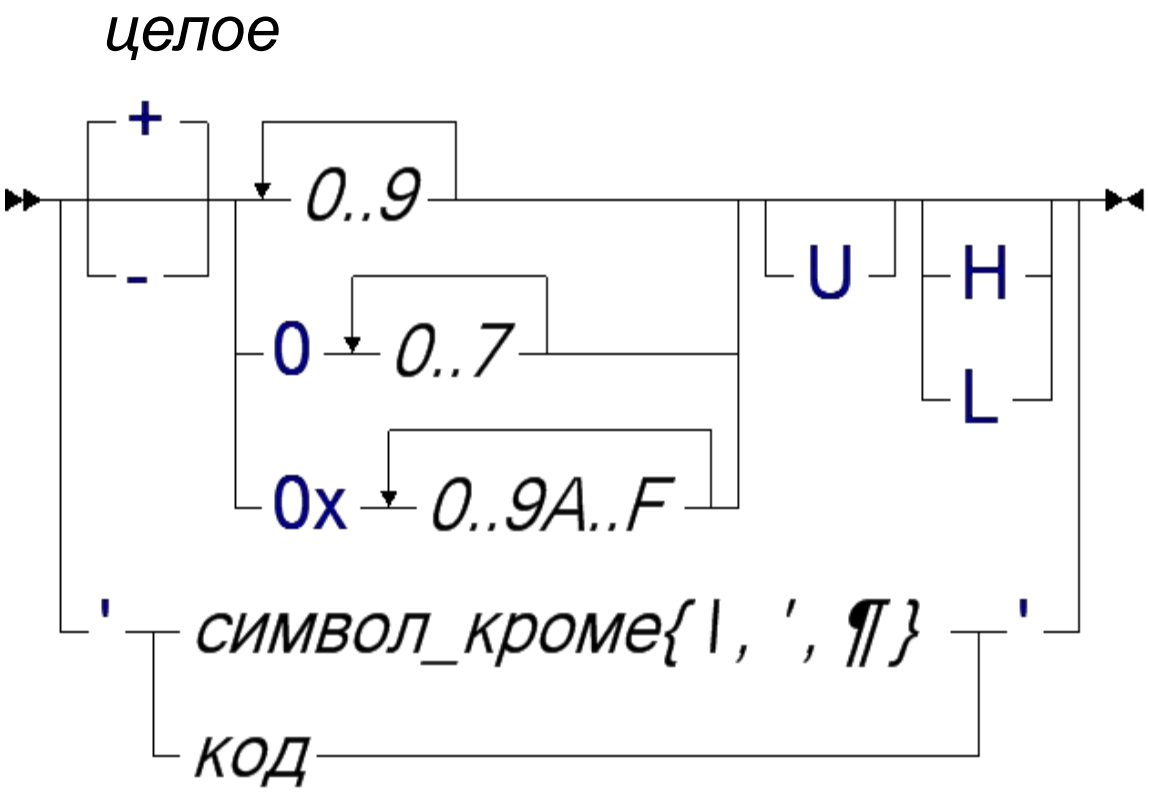
Целые – синтаксис и диапазон значений (C)

	Разряд-ность	unsigned	signed
char	8	0..255	-128..127
(short) int	16	0 .. 65,535	-32,768 .. 32,767
long int	32	0 .. 4,294,967,295	-2,147,483,648 .. 2,147,483,647

Algol-68: long long long int

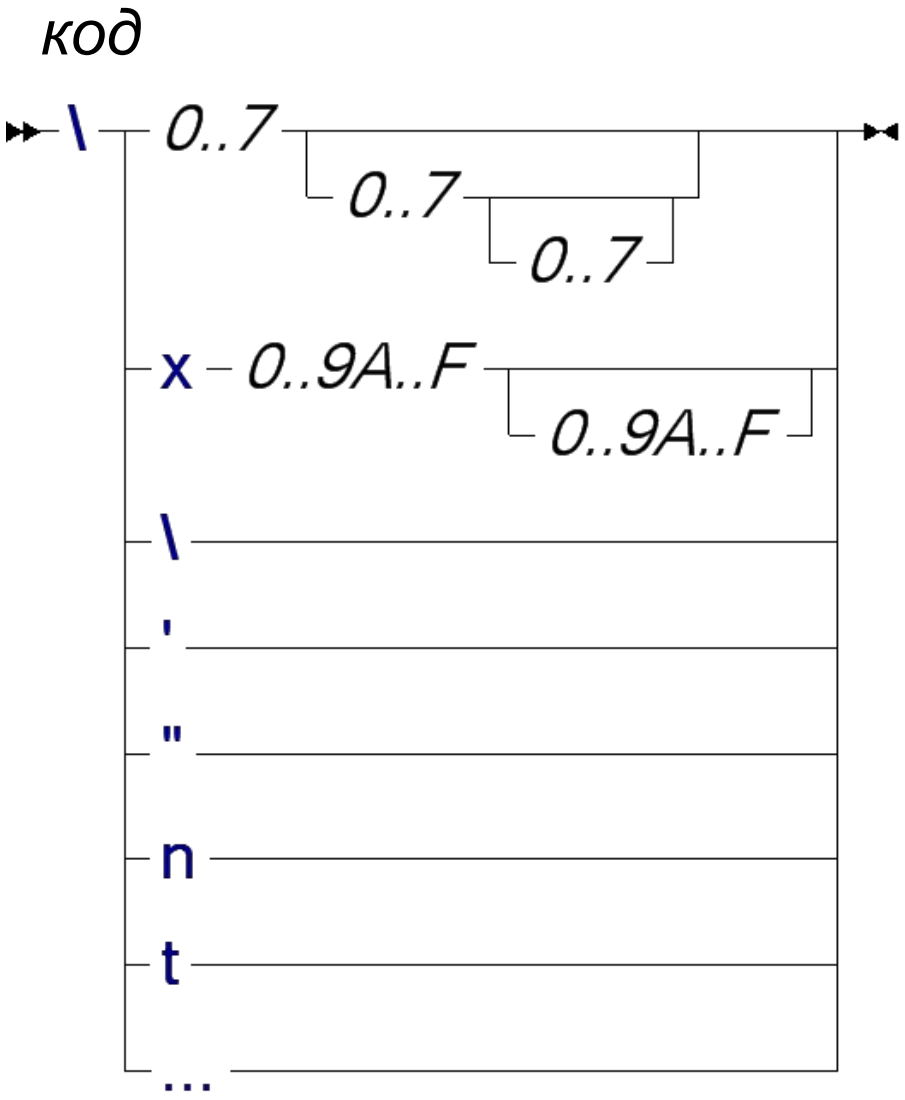
Целые – константы (C)

- *0..9* – десятичная цифра
- *0..7* – восьмиричная цифра
- *0..9A..F* – шестнадцатерич-ная цифра
- H – short
- L – long
- U – unsigned



Символы-коды (C)

- *0..7* – восьмиричная цифра
- *0..9A..F* – шестнадцатеричная цифра



Символы, как целые (C)

Символ – изображение своего кода:

‘\123’ == 0123

Пример: i-ая буква

– Pascal: chr(ord(‘A’) + i – 1)

– C: ‘A’ + i -1

Целые, как логические (С)

- 0 – ложь
- Не ноль – истина
- Операции
  - && - конъюнкция (и)
  - || - дизъюнкция (или)
  - ! - отрицание (не)

Пример:

- !1 || ‘A’ && 0x12L - результат = 1
- ‘\0’ || (‘A’ == ‘B’) – результат = 0

Целые, как битовые шкалы (C)

- & - побитовая конъюнкция
- | - побитовая дизъюнкция
- ^ - побитовый xor (неравенство)
- ~ побитовое отрицание
- <<, >> - сдвиги влево и вправо

Целые, как битовые шкалы (C)

Реализация операций над множествами

- set of 1..32 - unsigned long int
- S1 * S2, S1 + S2, S1 – S2

  S1 & S2, S1 | S2, S1 & ~S2
- x in S

  (1 << (x-1)) & S
- S1 <= S2

  S1 & S2 == S1
- [x..y]

  (y >= x ? ((1<<(y–x+1)) - 1) << (x-1) : 0)

Перечисление как целые

- Определение констант

```
#define red 0
#define green 1
#define blue 2
```

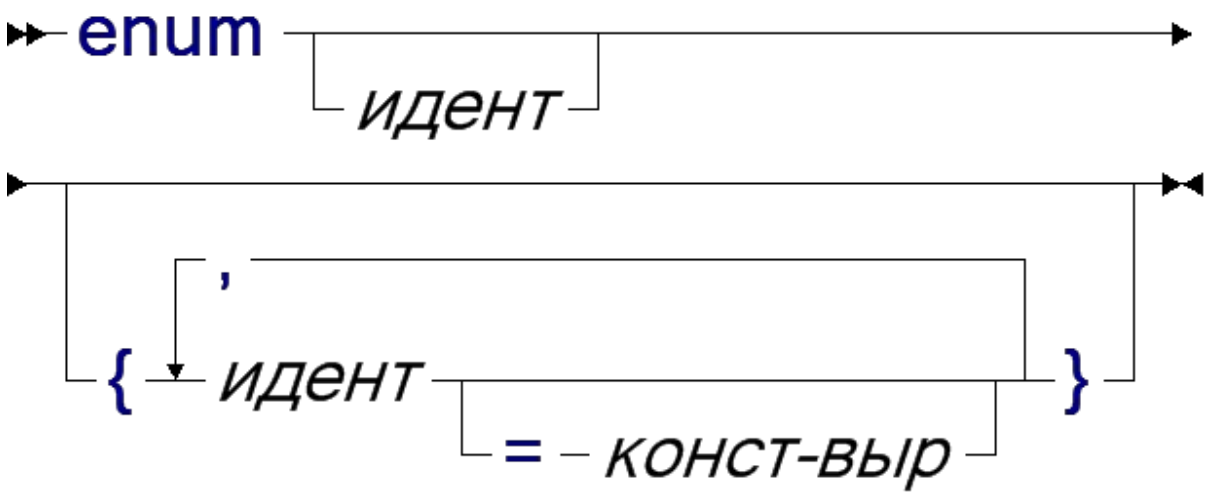
или

```
const int red = 0;
const int green = 1;
const int blue  = 2;
```

- Недостаток – лишняя информация
  - При добавлении новой константы – перенумеровать
  - Нестрогая типизация: blue / green, red + 8

Перечисление

- Синтаксис



- Пример:
  - **enum** StreetColor (red, green, blue)
  - **enum** WeekDay (
    Mon=1, Tue, Wed, Thu, Fri, Sat, Sun
)

Вещественные – представление 1

С фиксированной точкой

$b_{n-1}\ b_{n-2}\ \ldots\ b_0$ – последовательность битов, n – разрядность, p – размер дробной части

- b_n - знак (1 – отрицательные)
- Абсолютная величина:

$$\sum_{i=0}^{n-1} b_i * 2^{(i-p)}$$

Вещественные – представление 1 (пример)

n=8, p=2

- 000000 00 = 0
- 001111 10 = 8+4+2+1+1/2 = 15.5
- 100001 00 = -(1) = -1
- 111111 11 = -(16+8+4+2+1+1/2+1/4) = 31.75

Вещественные – представление 2

С плавающей точкой

- $b_{n-1}\, b_{n-2} \ldots b_p b_{p-1} \ldots b_0$ – последовательность битов,
- n – разрядность,
- p – размер мантиссы
- s – смещение порядка
- b_{n-1} - знак (1 – отрицательные)
- Абсолютная величина: $2^e * (1 + \sum_{i=0}^{p-1} b_i * 2^{(i-p)})$
- Порядок e = $(\sum_{i=p}^{n-2} b_i * 2^{(i-p)}) - s$

Вещественные – представление 2 (пример)

n=8, p=2, s=3

- 0 000 0000 = 0 (особый случай)
- 0 001 1110 = 2^{1-3} * (1+0.875) = 0.46875
- 1 000 0100 = -(2^{0-3} * (1+0.25)) = - 0.03125
- 1 111 1111 = -(2^{7-3} * (1+0.9375)) = 31

Вещественные – синтаксис и диапазон значений

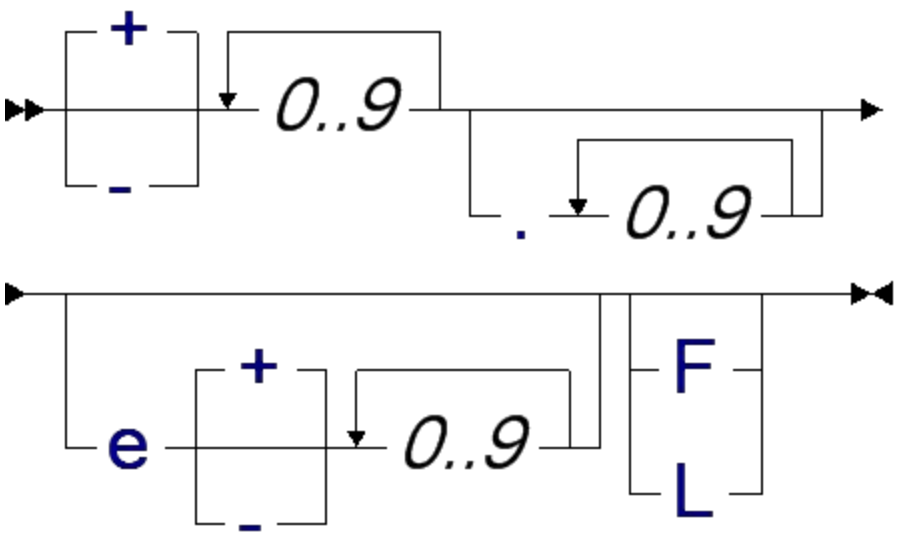
	Разряд-ность	Размер порядка	Сдвиг порядка	Размер мантиссы
float	32	8	127	23
double	64	11	1023	112

Вещественные – другие представления

- Неограниченная точность: неограниченный размер.
- Рациональные числа: числитель и знаменатель.
- Символьный: 2*sin(pi/6).
- Сумма (бесконечного) ряда, непрерывные дроби

Вещественные – константы (C)

- *0..9* – десятичная цифра
- F – short
- L – Double
- Неточность:
  - 12L – long int
  - 12.0L – double
  - 12e-5L - double



Вещественные – потеря точности

- С фиксированной точкой

  122.55 / 2 * 2 = 122.50

- С плавающей точкой
  - Большое + маленькое

    1.0e+38 + 1.0e-45f = 1.e+38
  - Преобразование системы счисления
    1.0e-40 = 9.999946E-41

Приведение типов

- Неявное – типы аргументов арифметической операции приводятся к максимальному
  - double
  - float
  - unsigned long
  - long
  - unsigned char
  - int
- Явное

  int x = 30000;

  x * 12 - переполнение, больше 32767

  (float) x * 12 == 360000f;

Указатели

- Описание (простой случай)

  T * p;

  T x;
- Операции
  - Взятие адреса

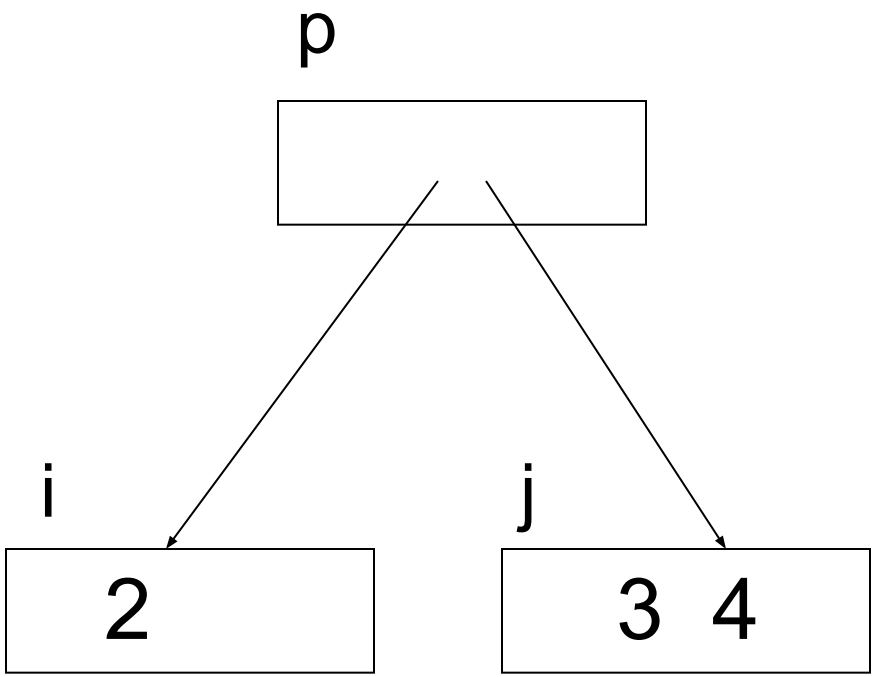
    p = &x
  - Разыменование – объект, на который указывает *p*

    *p
  - Свойства: &(*p) эквивалентно p, *(&x) эквивалентно x
- Литеральная константа: NULL – пустой указатель
- Реализация: адрес (ссылка, номер ячейки) указуемого объекта.

Указатели - пример

```
int i, j;
int * p;

p = &i;
*p =  2;
j = *p + 1;
p = &j;
*p = *p +1;
```



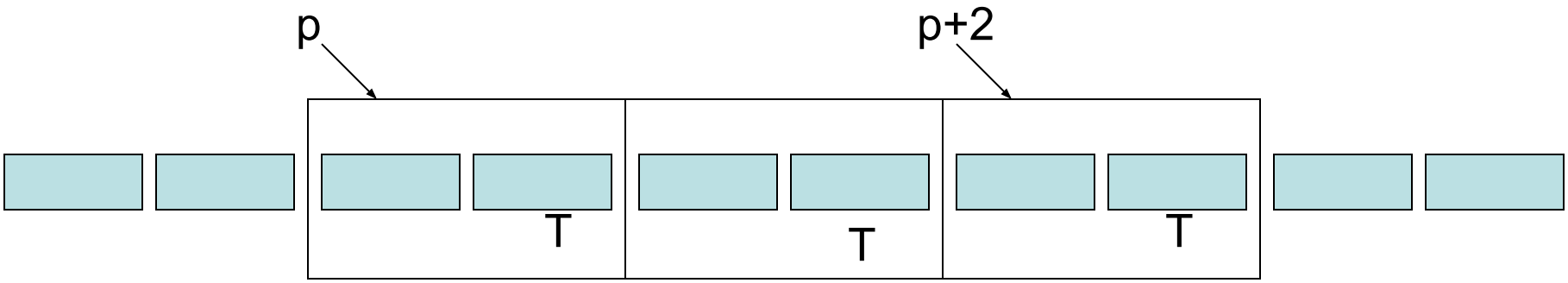
Адресная арифметика

Пусть

- *p* – указатель на объект типа *T*
- *p* начинается с байта с номером *a*
- размер *T* равен *s*

тогда *p+k* - указатель на объект типа T, который начинается с адреса *a* + *s*k*

(Аналогично p-k)



Адресная арифметика

- Указатели – (частично-)упорядоченный тип: порядок определён, только для указателей полученных из одного и того же указателя
- Пусть p1, p2 – однотипные указатели, k – целое, тогда
  - p1 + k – допустимо
  - k + p1 – недопустимо
  - p1 < p2 – допустимо
  - p1 – p2 – допустимо, результат целое
  - p1 + p2 - недопустимо

Тип void *

Указатель на «нечто» - можно явно привести к любому типу указателя.

Пример:

```
extern void * malloc(int c);
float * A = (float *) malloc(1000);
```

Массивы (C)

- Описание: (простой случай) T-тип размера s, N-константа

  T A[N]

  – Отведение непрерывного участка памяти размера N*s байт
  – A – константный указатель на первый элемент
- Операция: выборка компоненты

  A[i] эквивалентно *(A+i)
- Следствия:
  – Все массивы нумеруются с нуля.
  – Индекс последнего элемента равен N-1
  – Нет контроля границ индексов

Массивы (C)

- Литеральные значения (только в инициализации)

```
int A[] = { 5, 4, 3, 2, 1};
float A[2][2] = { {5, 4.0} , {3 , 2+2} };
```

Многомерные массивы

Pascal:

```
var A :
 array[1..N, 1..M]
 of real;
A[i,j]
```

C

- float A[N][M];

  A[i-1][j-1]
- float A[N*M];

  A[(i-1) * M + (j-1)]
- float A[N,M];

  A[i,j]

  – типичная ошибка

Динамические массивы

Размер определяется в процессе вычислений

Algol-60

```
integer N;
Read Int(N);
begin
    real array Data[1:N];
          …
end
```

C

```
int N;
scanf(“%d”,&N);
{
    float * Data =
     (float *)
       calloc(N,sizeof(float));
            …
   free(Data);
}
```

Динамические массивы

Размер массива – часть его представления

Modula-2

```
PROCEDURE
   Sum(A  : ARRAY OF
CARDINAL)
      : CARDINAL;
   …
BEGIN …
  FOR i := 0 TO HIGH(A) DO
    S := S + A[i];
  END;
  RETURN S;
END Sum;
```

C

```
unsigned long
    Sum(unsigned long A[],
      int N)
…

{
    for (i=0; i<N; i++)
        S = S + A[i];

    return S;
}
```

Подвижные массивы

Размер может меняться в процессе вычислений

Visual Basic

```
Input x
Cnt = Cnt + 1
If UBound(A) < Cnt Then
    Redim Preserve _
        A(0 To UBound(A) + 1000)
As Single
End If
A(Cnt) = x;
```

C

```
scanf("%f", &x);
Cnt = Cnt + 1;
If (SizeA < Cnt)
{
    SizeA = SizeA + 1000;
    A = (float*) realloc(A,
                SizeA * sizeof(float));
}
A[Cnt] = x;
```

C#

```
List <float> A;
float x = float.Parse(
      Console.ReadLine());
A.Add(x);
```

Подвижные массивы

Размер может меняться в процессе вычислений

C#

```
A.RemoveAt(i);
```

C

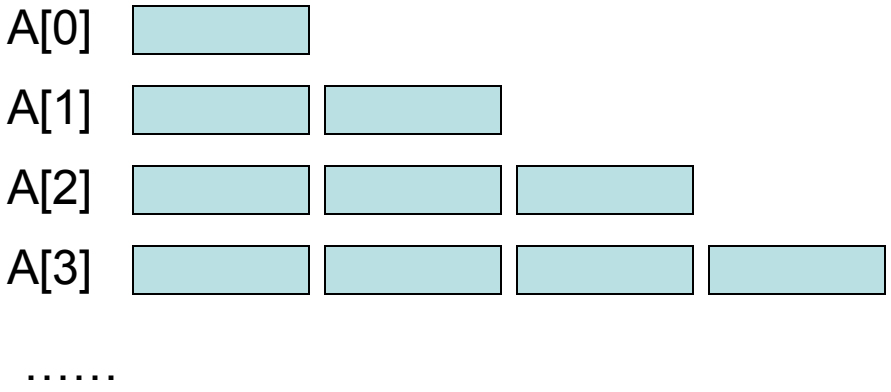
```
Memcpy(
    &(A[i]),
    &(A[i+1]),
    (SizeA - i -1) * sizeof(float));
Cnt = Cnt - 1;
```

Непрямоугольные массивы

Пример: треугольная матрица (C)

```
float * A[N];
for (i=0; i<N; i++)
   A = (float*) malloc(
       (i+1) * sizeof(float));

A[k][m]
```



Массивы-дескрипторы (Автокод Эльбрус)

Подмассив базового массива M2:

- Транспонированная матрица

  КОНСТ M2T = **ФОРМАВМ** ([i,j] = M2[j,i])
- Первая строка матрицы

  КОНСТ M1СТР = **ФОРМАВМ** ([j] = M2[1,j])
- Минор

  КОНСТ МИНОР = **ФОРМАВМ** ([i,j] = M2[i=0:K,j=:L])
- Диагонали

  КОНСТ ДМК = **ФОРМАВМ** ([i] = M2[i,i])

  КОНСТ ПМК = **ФОРМАВМ** ([i] = M2[i,ЧИТАТР(M2,1, ДЛИНИЗМ) - 1 - i])

Операции с массивами (Альфа)

- **массив** A[1:N,1:M], B[1:M,1:K], X, Y[1:N], Z[1:M]
- **вещественный** C

X*Y	Скалярное произведение
X+Y	Сумма векторов
X*C	Произведение вектора на скаляр
A*X	Произведение матрицы на вектор
Z*A	Произведение вектора на матрицу
A*B	Произведение матриц
A*C	Произведение матрицы на скаляр
A+A	Сумма векторов

Операции над массивами (Альфа)

- Больше, чем перегрузка операций (Algol-68, C++) – статическая проверка соответствия границ.
- Промежуточные массивы размещает сам транслятор (сравните с C)

  (A * B) * X + (2 * Y)
- Оптимизация, эффективные (параллельные) алгоритмы

Операции над массивами (APL)

- APL – A Programming Language язык, ориентированные на обработку структурных данных
- Богатый набор операци на массивами:
  - сдвиг, перестановка, сжатие, выбор индексов вхождений, транспонирование, упорядочение, …
- Распространение всех элементарных операций на массивы
- Оператор редукции

Операции над массивами (APL)

Пример: вычислить полином степени n от *x*, заданный массивом коэффициентов *A:*

/+ (A * (x ⍴ (n+1)))

/+ (A * (x ⍴ (0 1 2 … n)))

/+ (A * (x^0 x^1 x^2 … x^n))

/+ (A_0*x^0 A_1*x^1 A_2*x^2 … A_n*x^n)

$A_0*x^0 + A_1*x^1 + A_2*x^2 + \ldots + A_n*x^n$

Строки (Pascal)

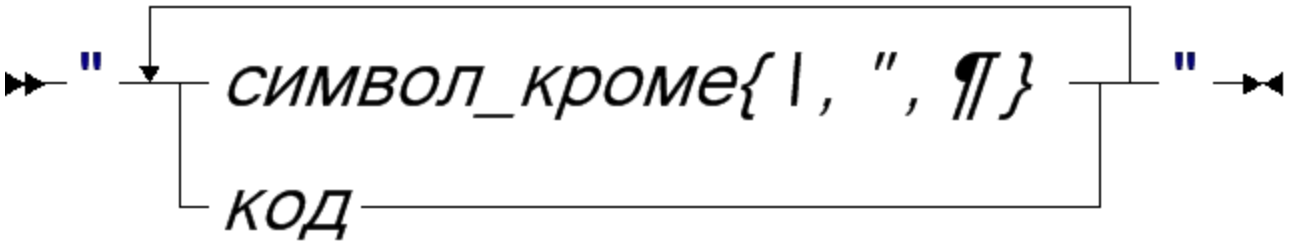
Категория	Последовательности символов длины не более 255
Синтаксис	**string** (**[** *целое* **]**)
Константы	‘’, ‘Hello, World’, ‘A’
Операции	s[i], Copy, Insert, Delete, Length, …
Реализация	первый байт – длина, остальные – символы строки

Строки как массивы (C)

Строка – указатель на последовательность символов, заканчивающуюся ‘\0’

unsigned char s[] = {‘H’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\0’};

Литеральное значение: “Hello, \”string\”!\n”



Строки как массивы (C)

Достоинства:

- Могут иметь произвольную длину
- Могут использоваться не только в инициализации (в отличии от других массивов)
- Не требуют специальных операций для доступа к содержимому

Строки как массивы (C)

Недостатки:

- Сложно определить длину (в отличии от Pascal)
- Все недостатки массивов
  - Нет контроля границ индексов
  - Нет подвижных массивов – память надо выделять «вручную»

Строки как массивы (C)

Операции <string.h>

- strlen(s) – длина s
- strcpy(s1,s2) – копирование строки
- strcat(s1,s2) – конкатенация строк
- strchr(s,c) – указатель на первое вхождение c в s
- …

Строки как массивы (С)

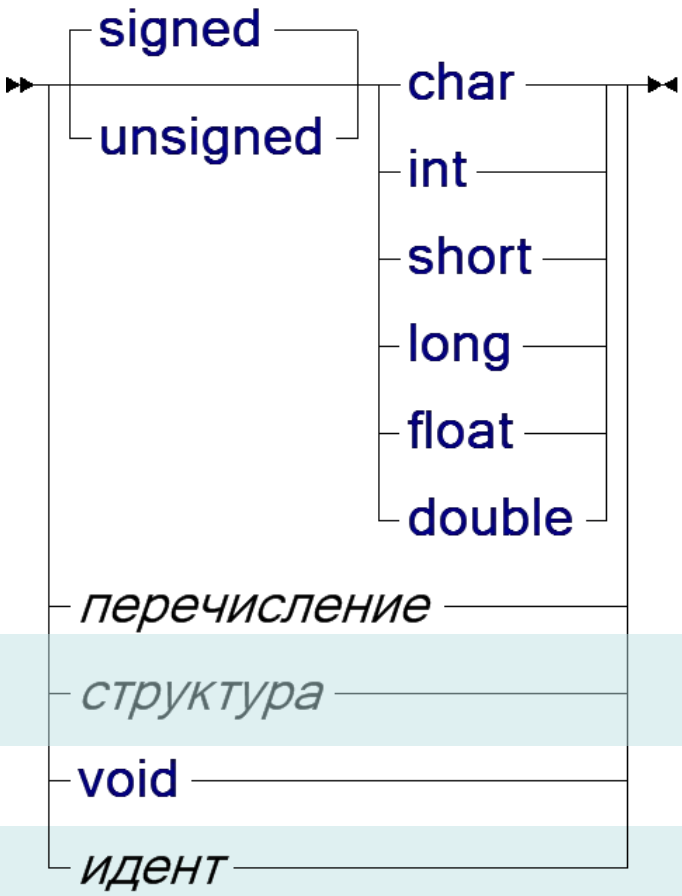
Пример (аналог Copy в Pascal – выборки подстроки)

```
unsigned char * PasCopy(unsigned char *source, int i, int l)
{
    unsigned char * dest = (unsigned char *) malloc(l+1);
    unsigned char * d = dest;
    unsigned char * s = &(source[i]);
    while ((*d++ = *s++) && l--)
      ;
    d[-1] = '\0';
    return dest;
}
```

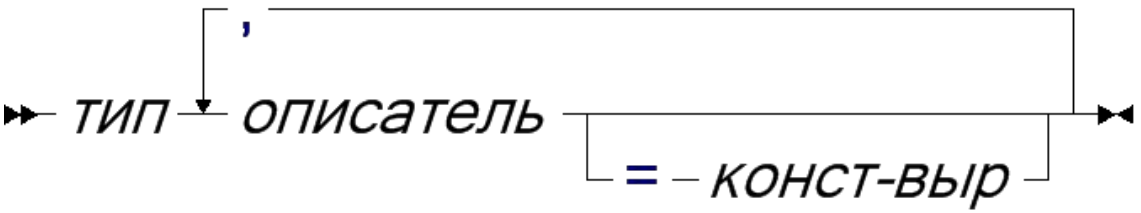
Описания

Синтаксис:

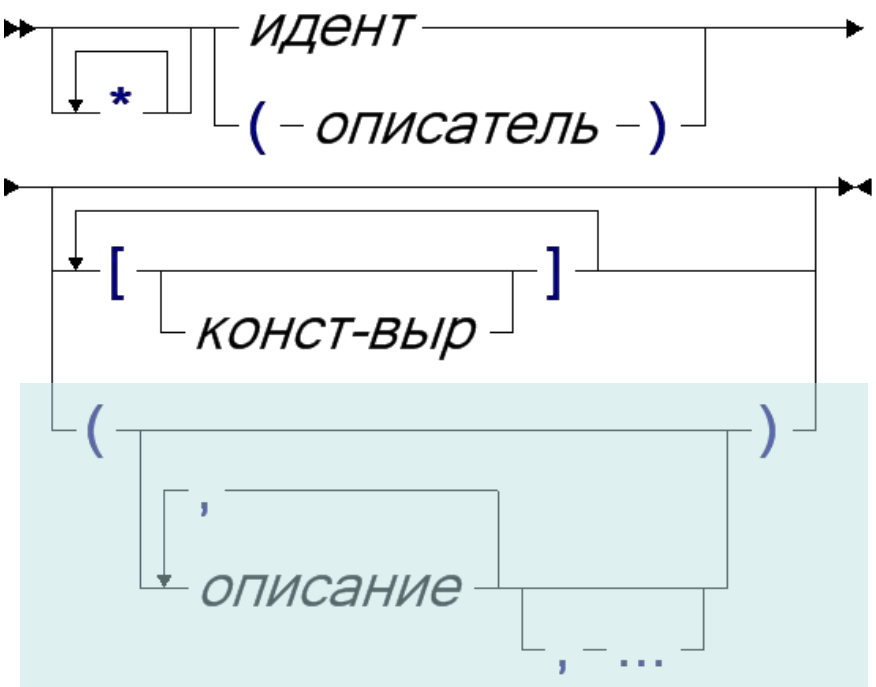
тип:



описание:



описатель:

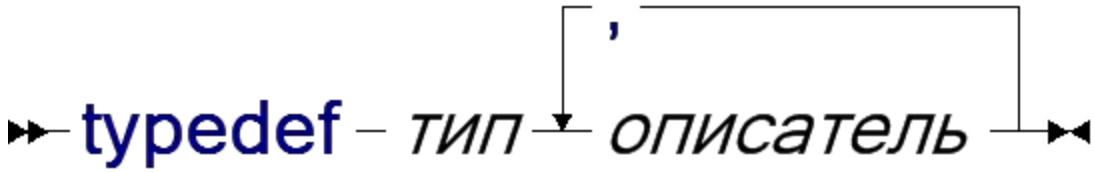


Описание - примеры

- Указатель на массив целых
  int (*x)[100]
- Массив из указателей на целые
  int * (x[100])
- Указатель на массив из массивов из указателей на указатели на перечисление WeekDay
  enum WeekDay ** ((*x)[][100])
- Указатель на целое, целое и целое равное 5
  long * p, n, m = 5

Описание типа

- Синтаксис



- Пример:

C

```
typedef float
Matrix[N][N];
Matrix A, *p;
```

Pascal

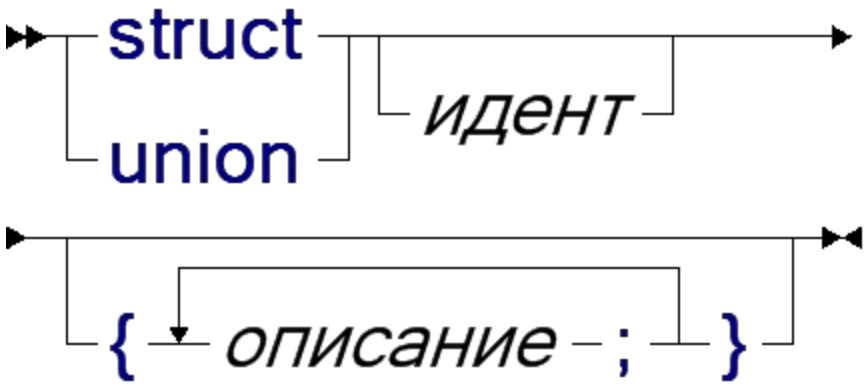
```
type Matrix =
     array[0..N-1]
       array [0..N-1]
         of real;
var A : Matrix;
      p : ^ Matrix;
```

Структуры

- Назначение: объединение разнотипных данных
  - **struct** – декартовое произведение
  - **union** – объединение
- Реализация:
  - **struct** – последовательное выстраивание
  - **union** – наложение различной разметки на участок памяти.

Структуры

- Синтаксис:



- Операции: **.** - выборка поля, например,

  S.code, A[i].re, (*p).next

  (последнее эквивалентно p->next)

Структуры

- Пример **struct**

```
typedef struct { re, im : float; } complex;
complex c1= {-1, 0}, c2 = {3.14,2.78}, c;
c.re = c1.re * c2.re – c1.im * c2.im;
c.im = c1.re * c2.im + c1.im * c2.re;
```

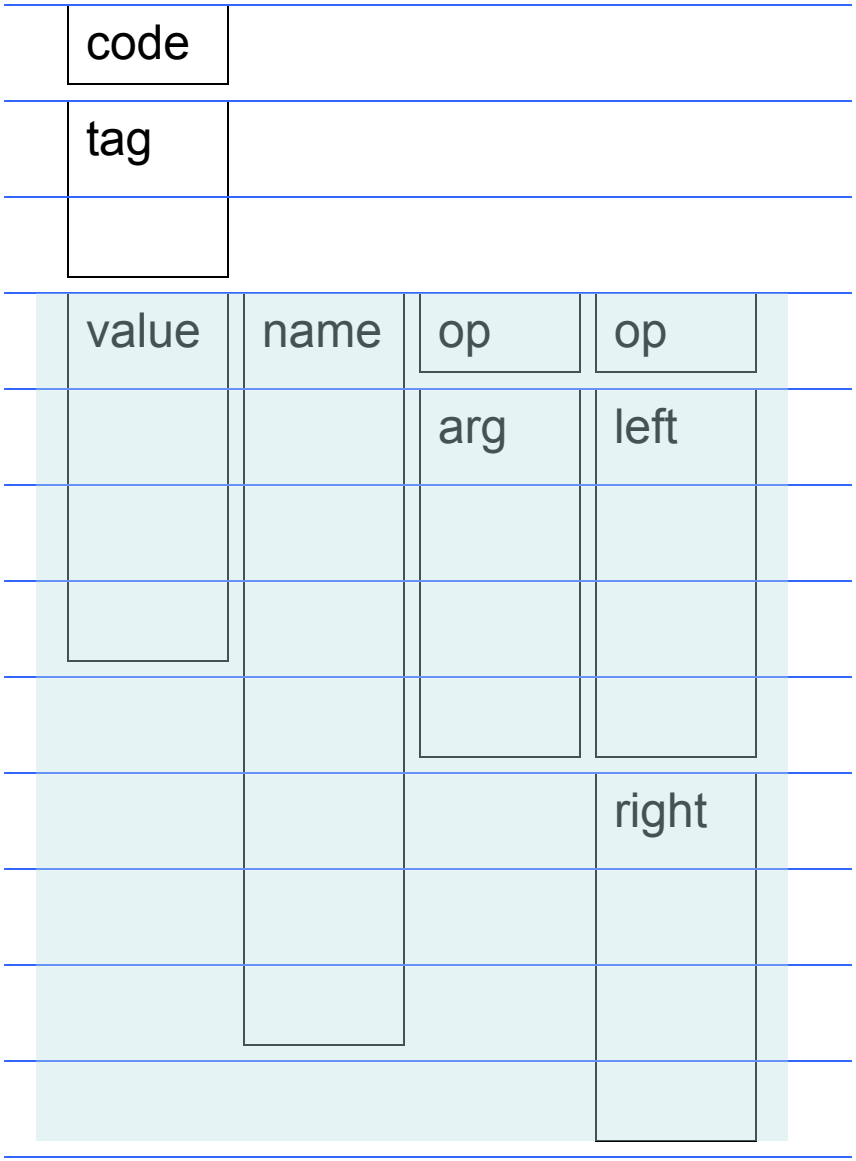
- Пример **union**

```
union { unsigned long l; unsigned char c[4];} b4;
b4.l = 0xAABBCCDD;
b4.c[1] = ‘A’;
```

(результат b4.l == 0xAA41CCDD)

Структуры - пример

```
struct expr {
  unsigned char code;
  int tag;
  union {
    float value;
    unsigned char name[8];
    struct {
      unsigned char op;
      struct expr * arg;
    } unop;
    struct {
      unsigned char op;
      struct expr * left, * right;
    } binop;
  } var;
} E;
```



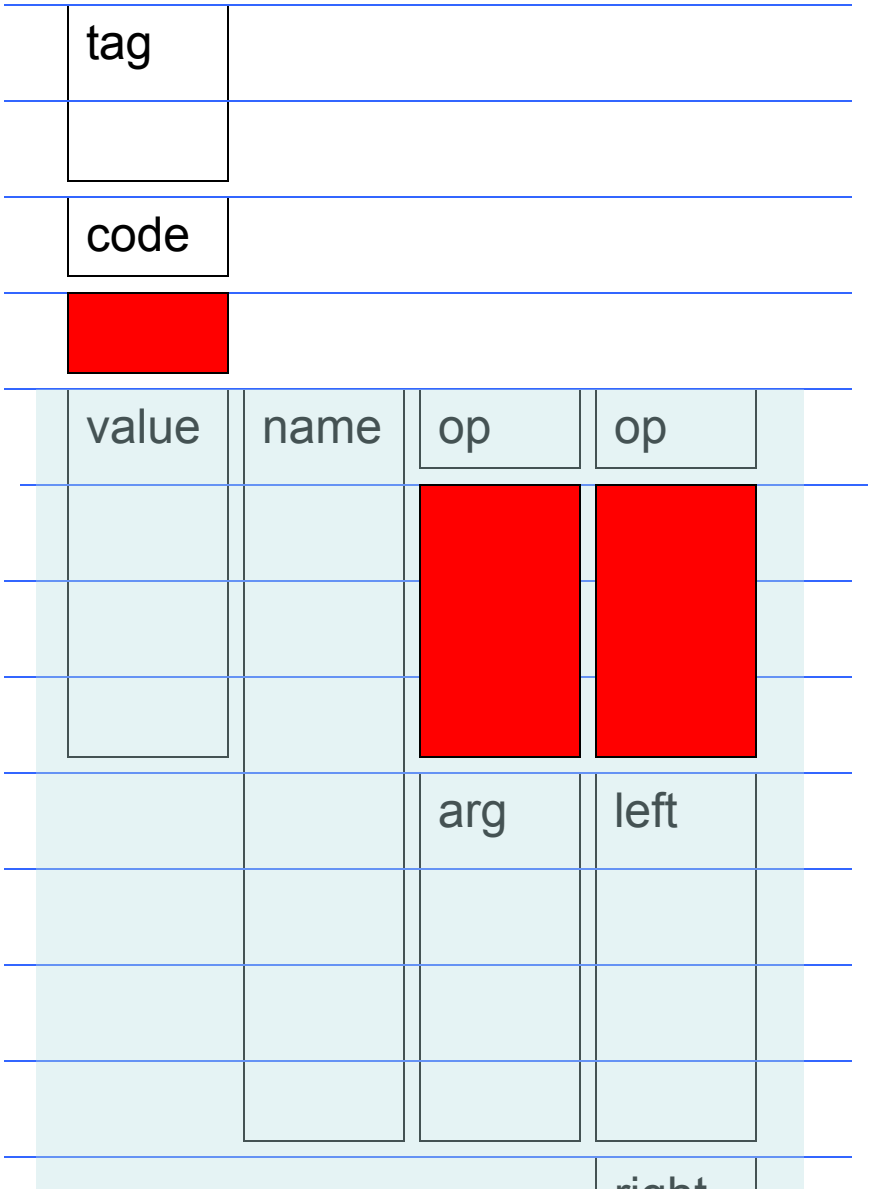
Структуры - пример

```
struct expr {
  int tag;
  unsigned char code;
  union {
    float value;
    unsigned char name[8];
    struct {
      unsigned char op;
      struct expr * arg;
    } unop;
    struct {
      unsigned char op;
      struct expr * left, * right;
    } binop;
  } var;
} * E;
```

```
type expr = ^ Sexpr;
    Sexpr = record
                    tag : integer;
            case code : char of
                  ‘v’ : (value : real);
                  ‘n’ : (name :
                    array[0..7] of char;)
                  ‘u’ : (unop : char;
              arg : expr; )
            ‘b’ : (binop : char;
              left, right : expr;)
            end;
var E : expr;
```

Структуры - выравнивание

```
struct expr {
  int tag;
  unsigned char code;
  union {
    float value;
    unsigned char name[8];
    struct {
      unsigned char op;
      struct expr * arg;
    } unop;
    struct {
      unsigned char op;
      struct expr * left, * right;
    } binop;
  } var;
} E;
```



union – «дыра» в контроле типов

```
typedef union
  {
     float r;
     int i;
     complex c;
     char * s
  } node;
node n;
n.s = "1234";
printf(“(%f,%f)”, n.c.re, n.c.im);
```

Algol-68:

```
mode node = union
        (real, int, compl, string);
node n := "1234";
case n in
  (real r): print(("real:", r)),
  (int i): print(("int:", i)),
  (compl c): print(("compl:", c)),
  (string s): print(("string:", s))
  out print(("?:", n))
esac
```

sizeof

- Размер типа данных или переменной
- Пример

```
char c, * p = “abc”,
   s[] = “abc”,
   a[3]={‘a’,’b’,’c’};
struct T{
   unsigned char code;
   struct T * left, * right;
};
```

c	1
p	4
s	4
a	3
struct T	12

sizeof

- Псевдооперация
  - Параметр – тип
  - Вычисление не требует вычисления аргумента, а только его типа
- Использования
  - динамическое размещение данных

```
Record * r = (Record*) malloc(sizeof(Record)),
       * a = (Record *) calloc(sizeof(*a),100);
```

  - копирование массивов

```
memcpy(dest, source, n * sizeof(*dest))
```

Присваивания

- Выражение с побочным эффектом (изменением состояния памяти)
  - *Получатель* (левая часть присваивания) – изменяемая переменная
  - *Источник* (правая часть присваивания) – присваиваемое значение
- Значение присваивания = присвоенное значение
- Приведение типов: тип источника не превосходит типа получателя.

Присваивание - пример

```
float A[N]; int  i, j;
A[i+j] = (i=(j=1)+2) + 4
```

1. Вычислить 1
2. Поместить 1 в j
3. К значению j прибавить 2
4. Поместить 3 в i
5. Вычислить 3+4 (результат 7)
6. Вычислить i+j
7. Вычислить элемент массива A[4]
8. Преобразовать 7 в вещественное 7.0
9. Поместить 7.0 в A[4]
10. Результат присваивания – 7.0

Присваивание – побочные эффекты!

- Если **вдруг** в предыдущем примере
  A[i+j] = (i=(j=1)+2) + 4
  сначала вычисляется получатель, то изменится A[0], а не A[4]
- Не специфицировано в каком порядке вычисляются операнды, например
  ((i=(j=2) + i) + (j=(i=1) + j)
  может быть равно как 5, так и 3.

Совмещенное присваивание

M[i+1] = M[i+1] + 2

эквивалентно
(при отсутсвии побочных эффектов)
M[i+1] += 2

- Сокращение записи – наглядность
- Соответствие смыслу – «увеличить M[i+1] на 2»
- Экономия вычислений – ячейка M[i+1] вычисляется лишь один раз
- Помимо += может быть -=, *=, /=, %=, &=, |=, ^=, <<=, >>=. (Но не может быть <=, &&=, !=)

Инкремент, декремент

- Префиксная форма

  ++ X эквивалентно X += 1

- Постфиксная форма

  X ++ эквивалентно (t = X, X+=1, t)

  1. Запомнить X во временной переменной
  2. Увеличить X на 1
  3. Выдать запомненное значение

- Аналогично для --

Совмещённое присваивание (пример)

(*p++) += 0x40

1. Запомнить значение указателя p
2. Извлечь значение символа *p
3. Прибавить к нему 0x40
4. Поместить полученное значение в *p
5. Взять запомненное на шаге 1значение указателя p
6. Увеличить его на 1
7. Поместить полученный указатель в p (перейти к следующему символу)

Путаница: = vs ==, & vs &&

- Присваивание встречается значительно чаще, чем сравнение на равенство (?)
- В системных программах & встречается чаще, чем && (?)
- Пример. Пусть x = 1, y = 2, тогда условие

  x=2 & x-y>0

  Реализуется как

  x = (((2 & x) – y) > 0),
  результат 0, побочно x=0