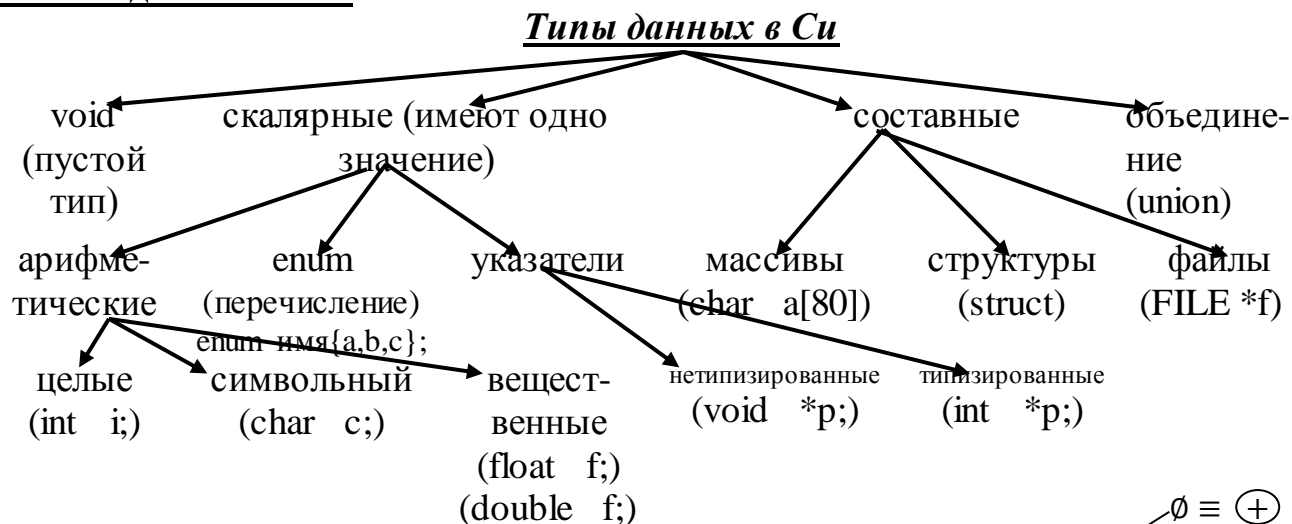


2 Типы данных в Си



Замечание:

старший бит числа интерпретируется как знаковый

кроме типов в Си есть еще модификаторы (спецификаторы) типа. Они уточняют внутреннее представление и диапазон значений стандартных типов (целых):

signed (со знаком) под целое число выделяется 2 байта или 4 (в зависимости от платформы)

- unsigned (без знака) целое число должно занимать в памяти не менее 4 байт или 8 (в зависимости от платформы)
- short (короткий)
- long (длинный).

Модификаторы типа записываются слева от основного типа.

unsigned int i; long v; ≡ long int v; ≡ signed long int v;

плохо для русских букв

Тип int является типом по умолчанию (в том числе для типа возвращаемого значения функции). В C++ уже нет типа данных по умолчанию.

2.1. Целые типы (пределы их изменений - в файле limits.h)

Паскаль			Си		
Имя типа	Размер	Диапазон	Имя типа	Размер	Диапазон
char	1 байт	chr(0-255)	unsigned char	1 байт	0 - 255
byte	1 байт	0 - 255	unsigned char	1 байт	0 - 255
			Замечания: 1) в Си не различаются символьный и целый типы. Переменную символьного типа можно инициализировать целым числом (char c=27;). Символьные константы и переменные можно использовать вместе в арифметических выражениях: d=c-'s'; 2) для представления символов кириллицы требуется unsigned char (по умолчанию signed char)		
shortint	1 байт	-128 .. +127	signed char	1 байт	-128 .. +127
			Замечание: можно явно не указывать signed. Он подразумевается по умолчанию		
integer	2 байта	-32763 - 32767	short (у констант суффикс H (или h))	всегда 2 байта	-32768..32767
			[signed] int	2 байта или 4 байта	-32769..32767
		он определяет представление значения в памяти	Замечание: по умолчанию размер int совпадает с размером слова на данной ЭВМ (на 16-разр. ЭВМ int=16 бит=short [int], на 32-разр. ЭВМ int= 32 бит = long [int])		
word	2 байта	0..65535	unsigned int	2 байта	0..65535
			у констант (целых) суффикс U : $32767+1 == 0$ $32767U +1 \neq 0$		
longint	4 байта	$-2^{31}..2^{31}-1$	long (\equiv signed long int)	4 байта	$-2^{31}..2^{31}-1$
			у констант суффикс L: 75L (такая константа будет занимать не 1 байт, как следовало бы из ее вида, а минимум 4)		
longword (в Delphi)	4 байта	0.. $2^{32}-1$ (0..4294967295)	unsigned long	4 байта	0.. $2^{32}-1$
			у констант суффикс UL: 125000UL		

В Си есть еще тип **long long**, но он не является стандартным) и поддерживается не всеми компиляторами), который занимает в памяти 8 байт. Для констант этого типа используется суффикс LL (или ll) и I64 (или i64), т.е. константы имеют вид 25LL или 25i64.

Также в Visual Studio есть типы (знаковые и беззнаковые):

_int8
 _int16
 _int32
 _int64

Для 2-байтовых символов в Си имеется тип `wchar_t` `cw` = `L'A`; символ занимает 2 байта
 переменная `cw` занимает 2 байта
`char` `c` = `'A'`; символ занимает 1 байт

Не во всех реализациях Си тип `wchar_t` является встроенным. Поэтому надо использовать

`#include <string.h>`


в котором типу `wchar_t` сопоставлен синоним `unsigned short`.

2.2. Вещественные типы (пределы их изменений - в файле `float.h`)

Паскаль			Си		
Имя типа	Размер	Диапазон	Имя типа	Размер	Диапазон
<code>real</code>	6 байт		нет	нет	нет
<code>single</code>	4 байта	3.4E-38 - 3.4E+38	<code>float</code> у констант суффикс <code>f</code> или <code>F</code>	4 байта	3.4E-38 - 3.4E+38
<code>double</code>	8 байт	1.7E-308 - 1.7E+308	<code>double</code> Константы с плавающей точкой <u>по умолчанию</u> имеют тип <code>double</code> . Можно явно указать другой (не <code>double</code>) тип константы с помощью суффиксов <code>F</code> и <code>f</code> (<code>float</code>) или <code>L</code> и <code>l</code> (<code>long</code>). Например, константа <code>2L</code> имеет тип <code>long double</code> , а <code>1.82F</code> – тип <code>float</code>	8 байт	1.7E-308 - 1.7E+308
<code>extended</code>	10 байт	3.4E-4932 - 3.4E+4932	<code>long double</code> у <u>констант</u> суффикс <code>L</code> или <code>l</code>	10 байт	3.4E-4932 - 3.4E+4932

в VS 8 байт, а в GCC(G++) 12 байт

2.3. Логический тип

Паскаль	Си
<code>boolean</code> (1 байт) <code>false</code> , <code>true</code>	<code>0</code> \equiv <code>false</code> , не ноль \equiv <code>true</code>
-----	----
логические значения	арифметические значения
	(<code>#define TRUE 1</code>)
	<code>#define FALSE 0</code> 

нельзя

Замечание 1

В Си нет логического типа данных: выражения, в которых требуются **логические значения**, интерпретируют значения "ноль" как `false` (ложь), а все другие (не равные нулю) как `"true"` (истина). Поэтому `if (25)` и `if (1)` неразличимы (с точки зрения будет или нет переход по условию «Да»), так как в скобках значение >0 . Обычной ошибкой является запись `if (a=25)...` \equiv `if (!0)....`

Замечание 2

В стандарте C++ есть **встроенный тип `bool`** со значениями `false` и `true` (занимают 1 байт, внутреннее представление `true` есть 1, а `false` - 0). В нестандартизованных версиях C++, как и в Си, встроенного логического типа не было, а вместо него в заголовочных файлах стандартной библиотеки был определен тип `BOOL` (через `typedef`) как псевдоним типа `int`, а значения логических констант `TRUE` и `FALSE` были определены через `#define` (см. выше).

`typedef unsigned char BOOL;`

2.4 Указатели

Паскаль	Си
Объявление	
v: pointer; a: ^integer; p: pchar;	void *v; ---- нетипизированный int *a; ---- типизированный char *p; ---- указатель на строку (с завершающим нулем)
Присваивание значения	
var b:integer; a:=@b; или a:=addr(b); b:=a^;	int b; a = &b; ---- взятие адреса b = *a; ---- разыменование
Адресная арифметика	
p := p + 1; --- можно для pchar a := a + 1; > нельзя для v := v + 1; обычного указателя	p = p + 1;---- если p - это типизированный указатель, то p++ означает переход к следующему элементу (не байту), который следует за тем, на который указывал указатель p (имеет смысл лишь в том случае, если p указывает на начало массива): int c[10]; p = &c; ≡ p = &c[0]; p = p+1; ≡ p+=1; ≡ p++; NB: при этом p должен быть объявлен так: int (*p)[10]; - указатель на массив (кстати int *p[10]; - массив указателей)

Примеры указателей

Паскаль	Си
Описание типов: type Intarray=array[1..10] of integer; Intptr = ^integer; Pnode = ^Node; <div style="border: 1px solid black; padding: 5px; display: inline-block;"> Node = record Data: integer; Next: Pnode; end; </div>	<div style="text-align: right;">объявление типа узла связанной динамической переменной</div> typedef int intarray[10]; typedef int *intptr; <div style="border: 1px solid black; padding: 5px; display: inline-block;"> typedef struct{ int data; node *next; } node, *pnode; </div> <div style="text-align: right;">указатель на тип стр-ры имя <u>типа</u> структуры</div>
Описание переменных: Var 1 i: integer; 2 IntPtr1 : IntPtr; 3 Intarray1: Intarray; 4 <pname> :^<тип>; 5 intptr: ^integer; 6 Buff1 :^Intarray; 7 Buff2 : array[0..10] of IntPtr; 8 PHead :^Node; 9 Head : Node;	int i; intptr intptr1; /* указатель на int */ intarray intarray1; /* массив целых */ <тип> *<pname>; /* типизир. указатель */ int *intptr; /* указатель на целое */ int (*buff1)[]; /*указатель на массив из int */ intarray *buff1; /* тоже указ. на массив из int */ int *buff2[]; /*массив указателей на целое*/ intptr buff2[10]; /*тоже массив указ. на целое */ node *phead;≡ pnode /*указатель на запись (указатель на узел типа node) */ node head ; // имя статической структуры // (не динамической) (типа node)
Применение указателей: 1 intptr1 := @I; 2 Phead := @Head; 3 Buff1 := @Intarray1; 4 Buff2[<u>1</u>] := IntPtr1; 5 <pname>^ := <значение>; 6 IntPtr1^ := 22; 7 Buff1^[<u>1</u>] := 0; 8 (Buff2[<u>1</u>])^ := 0; 9 Head.Data := 0; 10 Head.Next^ := 25; 11 PHead^.Next := nil; 12 head.next := nil;	intptr1 = &i; /* Инициализация указателя */ phead = &head; buff1 = &intarray1[0]; или buff1 = intarray1; buff2[<u>0</u>] = intptr1; *<pname> = <значение>; /* Обобщенная запись разыменования */ *intptr1 = 22; /* Разыменование указателя */ (*buff1)[<u>0</u>] = 0; // buff1 - указатель на массив //buff2 - массив указателей на целое *((buff2)[<u>0</u>]) = 0; константа head.data = 0; описана в stdlib.h как 0 *(head.next) = 25; (*phead).next= <u>NULL</u> ; или phead->next = NULL;*/ head.next = NULL; ≡ (*phead).next или 0

2.5. Перечисление

Паскаль

```
a: (b, c, d);
```

```
  | | |
  0 1 2
```

Си

элемент с явно получает значение

```
enum a {b, c=3, d};
```

```
      | | |
имя шаблона  0 3 4
```

```
enum a g;
```

```
//g может принимать
```

```
// значения b, c или d
```

Замечание: в отличие от Паскаля в языке Си можно каждому из перечисляемых имен явно присваивать значение. При этом тем именам, которые располагаются справа от последнего из явно получивших значение, будут присвоены следующие по порядку значения (на единицу больше, чем у соседа слева).

Пример: с помощью enum константы false и true булевский тип (шаблон) можно определить так:

```
enum boolean {FALSE, TRUE};
```

0	1

```
enum BOOLEAN b;
```

С помощью этого шаблона можно определить переменную

```
enum boolean a = TRUE;
```

//при этом нельзя присвоить a = 1, хотя a ≡ 1

Замечание: здесь булевский тип определен не полностью - только возможные значения. Любой тип определяет следующие стороны:

- | | |
|-----------------------------------|-----------------------|
| а) диапазон возможных значений; | наш шпблон |
| б) формат хранения в памяти; | определяет только это |
| в) множество допустимых действий. | |

2.6. Структурированные типы

2.6.1 Структуры и массивы структур

В Си к сожалению нет аналога оператора with ...do... .

Паскаль	Си
<pre> Var a:record b:char; c:integer; end;</pre>	<pre> struct { char b; int c; } <u>a</u>;</pre> <p style="text-align: center;">имя переменной-структуры</p>

При необходимости описания (выделения памяти) **нескольких однотипных структур** надо:

<pre> Type t=record b: integer; c: char; end;</pre>	<pre> typedef struct { int b; char c; } <u>t</u>;</pre> <p style="text-align: center;">имя типа структуры</p>	<p>или почти то же (без typedef):</p> <pre> struct { int b; char c; } V1, V2, V3;</pre> <p>Здесь нельзя объявить указатель того же типа, что на тип V1, V2 и V3</p>
<pre> Var V1,V2,V3: t; p: ^t;</pre>	<pre> t V1,V2,V3; t *p; //типизированный указатель на тип струк- //туры (для передачи структуры по адресу)</pre>	

Обращение к полям структуры без использования указателей:

<pre> begin v1.b := 25; ... end.</pre>	<pre> void main(void) { <u>V1</u>.b=25; ...}</pre>
--	--

Обращение к полям структуры с использованием указателей:

<pre> Begin p := @v1; p^b := 25;</pre> <div style="display: inline-block; vertical-align: middle;"> </div>	<pre> void main(void) { p = &v1; (*p).b=25; // 1-й путь p->b=25; // 2-й путь}</pre>
--	--

Необходимость использования указателей на структуры вместо имен самих структур связана с **передачей структур в функции**. Если структуру большого размера передавать в функцию целиком по значению, то это будет связано с дополнительными затратами времени и места на перенос копии структуры в стек и из стека при вызове функции и возврата из нее. Поэтому для сокращения времени передачи структур как параметров функций надо передавать их по адресу. Для этого надо сделать следующее:

- 1) в заголовке функции, куда надо передать структуру, должен быть **объявлен указатель** на тип структуры;
- 2) при вызове этой функции в качестве фактического параметра для указателя на структуру надо **передавать адрес** соответствующей структуры;
- 3) в теле функции (куда передали структуру) надо использовать **разыменование**.

Рассмотрим **пример**:

```
typedef struct
{
    int a;
    char b;
} t;
t v1; //передаваемая в функцию (процедуру) переменная - структура v1 типа t
void f(t *ptr) //процедура, в которую передается указатель на структуру типа t
{
    ptr->a = 1; //обращение к элементу структуры через указатель
или
    (*ptr).a = 1; //обращение к элементу структуры через указатель
}
void main(void)
{
    f(&v1);
}
```

① имя типа структуры

② вызов процедуры с передачей структуры по адресу

③

И в языке Си, и в Паскале можно описывать **массивы структур** (таблицы):

<p>Type</p> <p>t = record</p> <p> a: integer;</p> <p> b: char;</p> <p>end;</p> <p>Var</p> <p> vs: array[1..25] of t;</p> <p> ps: ^t;</p> <p>begin</p> <p> ps := @vs; или ps := vs;</p> <p> vs[6].a := 25;</p> <p>или нельзя в Паскале</p> <p> (ps+5)^.a := 25;</p>	<pre>typedef struct { int a; char b; } t; t vs[25]; //массив структур типа str t *ps; //указатель на тип структуры void main(void) { ps = vs; или ps=&vs[0]; или ps:= &vs; vs[5].a=25; /*обращение к элементу структуры */ или в Си индексы начинаются с 0 (*(ps+5)).a=25; или (ps+5) ->a =25;</pre>
---	---

имя типа структуры

имя массива есть константный указатель

В программах на Си при *описания связанных структур* (стек, список и т.п.) приходится использовать внутри структуры указатель на тип этой же структуры – причем этот тип определяется ниже по тексту программы):

<pre>Type lst_ptr=^node; node=record inf: integer; link:lst_ptr; end; Var head: lst_ptr;</pre>	<pre>typedef struct { int inf; link - это указатель на тип структуры node *link; /*лучше pnode link; */ } node, *pnode; pnode *head; //head – это указатель на тип // структуры (как и поле link)</pre>
--	--

Особенность: имя типа структуры в старом Си **нельзя** было использовать в качестве имени типа внутри этой же самой структуры

Поэтому для описания вложенных структур в старом Си используют другой способ объявления однотипных структур – с помощью так называемых *шаблонов структур* (память под шаблоны не выделяется).

Рассмотрим пример шаблона:

<pre>Type t=record b: integer; c: char; end; Var V1,V2,V3: t;</pre>	<pre>struct t { int b; char c; }; ... struct t V1,V2,V3;</pre> <p>имя шаблона структуры</p> <p>имена переменных-структур</p> <p>в C++ здесь указывать слово struct уже не надо</p>
---	---

Можно еще так:

```
struct t
{
  int b;
  char c;
};
```

Здесь создается и шаблон с именем t (без выделения под него памяти) и переменная с именем v (с выделением памяти).

С использованием шаблона описание рассмотренной выше вложенной структуры будет выглядеть так:

```
typedef //здесь не нужен typedef
struct node
{
    int inf;
    struct node *link;
};
struct node *head;
```

имя шаблона

переменная-указатель на структуру

Если структура глобальная или определена с описателем (классом памяти в Си) *static* в теле функции, то ее можно **инициализировать в месте описания**. Начальные значения для каждой структуры задаются внутри набора фигурных скобок (в Паскале были круглые скобки). **Соответствие** между константами и полями структуры устанавливается **по порядку записи** (первая по порядку константа связывается с первым по порядку полем и т.д.). Поля, которым начальные значения явно не присвоены, будут заполнены нулями.

Пример инициализации структуры в месте ее описания:

```
struct
{
    int a;
    int b;
} s1 = {1, 2},
  s2 = {3};
```

s1.a == 1
s1.b == 2
s2.a == 3
s2.b == 0

11

Структуры могут использоваться для обеспечения доступа к отдельным битам слова. В этом случае компонентами структуры являются **битовые поля**.

Битовое поле может быть любого целого типа (char, int, short, long) - как знакового так и беззнакового. Обычно битовое поле имеет длину до 16 бит в 16-разрядной среде и до 32 бит в 32-разрядной среде. Поля могут иметь имена, а могут и не иметь (если их не планируется использовать). **Поля размещаются в памяти** последовательно так, что направление возрастания адресов (номеров разрядов - у младшего разряда номер меньше) соответствует порядку описания полей.

Пример структуры с битовыми полми:

```
struct
{
    +--- имя не обязательно
    +-----+-----+
    мл. | [unsigned имя1] | : длина_битового_поля1;
        | [unsigned имя2] | : длина_битового_поля2;
        | .....         |
    ст. | [unsigned имяN] | : длина_битового_поляN;
    +-----+-----+
} имя;
```

в битах

направление возрастания адресов (номеров разрядов)

константа, заданная выражением целого типа, указывающая сколько битов отведено битовому полю имя1.

При хранении в памяти структуры могут «выравниваться» (aligned) компилятором на границу слова (по умолчанию) или на границу байта.

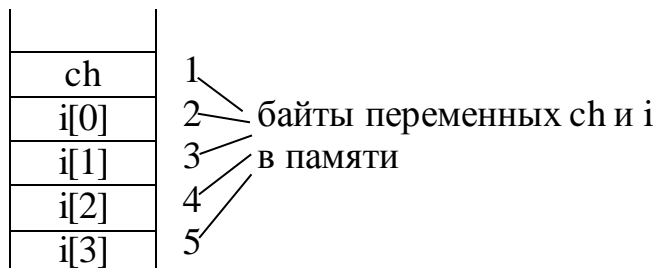
Выравнивание структур в памяти:

а) на границе байта

При выравнивании структурной переменной **на границе байта** отдельные ее поля в памяти располагаются без лишних "зазоров" между полями. Начинаться переменная будет с любого (четного или нечетного) адреса, ее длина равна сумме длин полей структуры.

struct

```
{  
  char ch; //1 байт  
  int i;   // 4 байта  
}str;
```



б) на границе слова (по умолчанию)

При выравнивании **на границе слова** (по умолчанию) компилятор при размещении структурной переменной в памяти вставляет между ее полями (или между элементами массива) зазоры (пустые байты) для того, чтобы соблюдались следующие **правила**:

1) отдельная структурная переменная (массива, структура) должна каждая начинаться на границе слова, т. е. с четного адреса;

2) элементы массива и поля структуры, тип которых не совпадает с типом char, каждое поле и элемент массива будут начинаться с четного адреса (имеет четное смещение от начала структурной переменной);

3) при необходимости в конце структурной переменной (массива, структуры) добавляется еще один байт так, чтобы число байтов структурной переменной было четным.

В процессе выравнивания по умолчанию поля выравниваются по **границе, кратной своему размеру**:

- 1-байтовые поля не выравниваются;
- 2-байтовые выравниваются по четным адресам;
- 4-байтовые выравниваются на позиции, кратные четырем;
- и т. д.

для 4-байтовых слов имеем:



Выравнивание сказывается **критическим образом на скорости переноса** структурных переменных между внешней и оперативной памятью (при выравнивании на границе слова – **быстро**).

В программе можно принудительно **установить выравнивание на границу байта**:

```
#pragma pack(push,1)
    Описание структуры
#pragma pack(pop)           // восстановление настройки по умолчанию
```

 **директива** (препроцессора, компилятора, линкера)

В отличие от опций среды или командной строки (они действуют на весь текст программы) директивы `#pragma` позволяют задавать установки по отношению к отдельным частям программы (функциям, переменным, ...).

2.6.2. Объединения

Аналог объединения в Паскале – **вариантная запись**, где имеется только вариантная часть и под варианты поля выделена одна и та же область памяти.

Паскаль	Си
<pre>Var a: record case boolean of true :(b:char); false:(c:integer); end;</pre>	<pre>union [имя шаблона] { <u>unsigned</u> char b; int c; (или short c;) } <u>a</u> ;</pre> <p style="text-align: center;">имя переменной</p>

Объединения применяют, если необходимо использовать **одну и ту же область памяти** для хранения **в разное время значений разных типов** для следующих целей:

- **минимизация** используемого объема памяти, если в каждый момент времени только один объект из многих, которые накладываются в памяти друг на друга, является **активным**;

- **интерпретация** частей основного представления объекта одного типа с помощью набора объектов другого типа.

Для работы с объединениями в Си используется тип `union` со следующей формой записи:

<pre>union { описание компонента1; описание компонента2; }</pre>	<p>Для каждого компонента выделяется одна и та же область памяти, т.е. компоненты перекрываются в памяти</p>	<p>Похоже на то, что было в Паскале:</p> <pre>Var Компонент1 : тип1; Компонент2 : тип2 absolute компонент1;</pre>
--	--	--

Объединения подобны структурам. Это значит, что доступ к компонентам объединения осуществляется тем же способом, что и к компонентам структуры.

Пример

```
union
{
  unsigned char i;
  struct
  {
    unsigned il :1;
    :6;
    unsigned ih :1;
  } s;
} u;
```

Здесь объявлена **структура с битовыми полями**. Для описания полей используется запись вида `имя : размер`. Размер задается в битах.

В данном случае структура занимает 8 бит. Нас интересуют 2 бита – самый старший и самый младший. Использование этой структуры в объединении с переменной `i` позволяет получать доступ к любому биту этой переменной.

Использование:

`u.i = 255; // изменяются все биты структуры (переменной i)`

`u.s.ih = 1; // изменяется старший бит переменной i`

имя структуры (части объединения)
имя объединения (union'a)